ULTRIX

Reference Pages for Unsupported Software

Order Number: AA-MF05B-TE
June 1990

Product Version: ULTRIX Version 4.0 or higher

This manual describes the unsupported commands, library routines, special files, file formats, games, and macro packages for RISC and VAX platforms.

digital equipment corporation
maynard, massachusetts

Acknowledgments

The software described in this documentation is based in part on the Fourth Berkeley Software Distribution under license from The Regents of the University of California. The following individuals and institutions are acknowledged for their role in its development:

The Electrical Engineering and Computer Science Department at the Berkeley Campus of the University of California, Eric Allman, Ken Arnold, Ozalp Babaoglu, Scott B. Baden, Jerry Berkman, John Breedlove, Earl T. Cohen, Robert P. Corbett, Mike Curry, Steve Feldman, Tom Ferrin, John Foderaro, Susan L. Graham, Charles Haley, Robert R. Henry, Andy Hertzfeld, Mark Horton, S.C. Johnson, William Joy, Howard Katseff, Peter Kessler, Jim Kleckner, J.E. Kulp, James Larus, Kevin Layer, Mike Lesk, Steve Levine, Jeff Levinsky, Louise Madrid, M. Kirk McKusick, Colin L. McMaster, Mikey Olson, Geoffrey Peck, Ed Pelegri-Llopart, Rob Pike, Dave Presotto, John F. Reiser, Asa Romberger, Bill Rowan, Jeff Schreibman, Eric P. Scott, Greg Shenaut, Eric Shienbrood, Kurt Shoens, Keith Sklower, Helge Skrivervik, Al Stanberger, Ken Thompson, Michael C. Toy, Richard Tuck, Bill Tuthill, Mike Urban, Edward Wang, David Wasley, Joseph Weizenbaum, Jon L. White, Glenn Wichman, Niklaus Wirth.

About Unsupported Reference Pages

The *Reference Pages for Unsupported Software* describe commands, routines, file formats, special files and games for RISC and VAX platforms that are part of the optionally installed, unsupported software subset. The unsupported software is not all documented and there are differences between the documentation levels on the RISC and VAX platforms. QARs are not accepted on the unsupported reference pages.

The unsupported reference pages are each labeled "Unsupported" under the title.

Sections

The reference pages for the unsupported software are in one binder that is divided into seven sections according to topic. Within each section, the reference pages are organized alphabetically by title, except Section 3, which is divided into subsections.

Some reference pages carry a one- to three-letter suffix after the section number, for example, `abort`(3f). The suffix indicates a "family" of reference pages for that utility or feature.

Following are the sections that make up the *Reference Pages for Unsupported Software.*

Section 1: Commands

This section describes unsupported commands that are available to all ULTRIX users.

Section 3: Library Routines

This section describes the unsupported routines available in ULTRIX libraries. Routines are sometimes referred to as subroutines or functions.

Section 4: Special Files

This section describes unsupported special files, related device driver functions, databases, and network support.

Section 5: File Formats

This section describes the format of unsupported system files and how the files are used.

Section 6: Games

The reference pages in this section describe the games that are available in the unsupported software subset.

Section 7: Macro Packages and Conventions

This section contains miscellaneous information, including ASCII character codes, mail addressing formats, text formatting macros, and a description of the root file system.

Section 8: Maintenance

This section describes unsupported commands for system operation and maintenance.

Platform Labels

The *Reference Pages for Unsupported Software* contain entries for both RISC and VAX platforms. Pages that have no platform label beside the title apply to both platforms. Reference pages that apply only to RISC platforms have a "RISC" label beside the title and the VAX-only reference pages that apply only to VAX platforms are likewise labeled with "VAX." If each platform has the same command, system call, routine, file format, or special file, but functions differently on the different platforms, both reference pages are included, with the RISC page first.

Reference Page Format

Each reference page follows the same general format. Common to all reference pages is a title consisting of the name of a command or a descriptive title, followed by a section number; for example, `date`(1). This title is used throughout the documentation set.

The headings in each reference page provide specific information. The standard headings are:

Name	Provides the name of the entry and gives a short description.
Syntax	Describes the command syntax or the routine definition. Section 5 reference pages do not use the Syntax heading.
Description	Provides a detailed description of the entry's features, usage, and syntax variations.
Options	Describes the command-line options.
Restrictions	Describes limitations or restrictions on the use of a command or routine.
Examples	Provides examples of how a command or routine is used.
Return Values	Describes the values returned by a system call or routine. Used in Sections 2 and 3 only.
Diagnostics	Describes diagnostic and error messages that can appear.
Files	Lists related files that are either a part of the command or used during execution.
Environment	Describes the operation of the system call or routine, with the POSIX and SYSTEM V environments. If the environment has no effect on the operation, this heading is not used. Used in Sections 2 and 3 only.

See Also	Lists related reference pages and documents in the ULTRIX documentation set.

Conventions

The following documentation conventions are used in the reference pages.

%	The default user prompt is your system name followed by a right angle bracket. In this manual, a percent sign (%) is used to represent this prompt.
#	A number sign is the default superuser prompt.
user input	This bold typeface is used in interactive examples to indicate typed user input.
system output	This typeface is used in text to indicate the exact name of a command, routine, partition, pathname, directory, or file. This typeface is also used in interactive examples to indicate system output and in code examples and other screen displays.
UPPERCASE lowercase	The ULTRIX system differentiates between lowercase and uppercase characters. Literal strings that appear in text, examples, syntax descriptions, and function definitions must be typed exactly as shown.
rlogin	This typeface is used for command names in the Syntax portion of the reference page to indicate that the command is entered exactly as shown. Options for commands are shown in bold wherever they appear.
filename	In examples, syntax descriptions, and routine definitions, italics are used to indicate variable values. In text, italics are used to give references to other documents.
[ ]	In syntax descriptions and routine definitions, brackets indicate items that are optional.
{ | }	In syntax descriptions and routine definitions, braces enclose lists from which one item must be chosen. Vertical bars are used to separate items.
. . .	In syntax descriptions and routine definitions, a horizontal ellipsis indicates that the preceding item can be repeated one or more times.
. . .	A vertical ellipsis indicates that a portion of an example that would normally be present is not shown.
cat(1)	Cross-references to the *ULTRIX Reference Pages* include the appropriate section number in parentheses. For example, a reference to cat(1) indicates that you can find the material on the cat command in Section 1 of the reference pages.

Online Reference Pages

The ULTRIX reference pages for unsupported software, as well as supported, are available online if installed by your system administrator. The `man` command is used to display the reference pages as follows:

To display the `troff`(1) reference page:

```
% man troff
```

To display the Name lines of all reference pages that contain the word "Fortran":

```
% man -k Fortran
```

Users on ULTRIX workstations can also display the reference pages using the unsupported `xman` utility, if installed. See the `xman`(1X) reference page for details.

Reference Pages for Supported Software

The reference pages for the supported ULTRIX software are in the document *ULTRIX Reference Pages*.

You can use the `setld` command to install all or some of the unsupported software subsets included with your ULTRIX. See `setld(8)` in the *ULTRIX Reference Pages* for further information.

Table 1: VAX Unsupported Subset Sizes

Subset Name	Kilobytes Used / (root)	/usr	/usr/var	Total
ULXAPL400	0	270	0	270
ULXBASE400	56	2558	1	2615
ULXBIB400	0	195	0	195
ULXCOURIER400	0	105	0	104
ULXCPM400	0	28	0	28
ULXDOC400	0	3448	0	3448
ULXEDIT400	0	6237	0	6237
ULXF77400	0	733	0	733
ULXGAMES400	0	2230	0	2230
ULXHYPER400	0	79	0	79
ULXICON400	0	347	0	347
ULXINGRES400	0	2638	0	2638
ULXLEARN400	0	653	0	653
ULXLISP400	0	3963	0	3963
ULXMAN400	0	319	0	319
ULXMOD2400	0	1036	0	1036
ULXNEWS400	0	866	0	866
ULXNOTES400	0	1179	0	1179
ULXRCS400	0	227	0	227
ULXSHELLS400	0	56	0	56
ULXSPMS400	0	1236	0	1236
ULXTOOLS400	0	55	0	55
ULXVARIAN400	0	2736	0	2736

Table 2: RISC Unsupported Subset Sizes

Subset Name	Kilobytes Used / (root)	/usr	/usr/var	Total
UDXBASE400	95	3614	1	3710
UDXBIB400	0	292	0	292
UDXCOURIER400	0	165	0	165
UDXDOC400	0	3448	0	3448
UDXEDIT400	0	6237	0	6237
UDXGAMES400	0	2507	0	2507
UDXLEARN400	0	734	0	734
UDXMAN400	0	129	0	129
UDXNEWS400	0	866	0	866
UDXNOTES400	0	1885	0	1885
UDXRCS400	0	213	0	213
UDXSHELLS400	0	95	0	95
UDXTOOLS400	0	112	0	112

Table 3 lists each unsupported software subset and gives a brief description of its contents. The Dependencies column shows the names of any other subsets and kernel configuration file options related to each subset.

Table 3: Unsupported Software Subset Descriptions and Dependencies

Subset Name	Contents	Dependencies
ULXAPL400*	APL Development Package User-contibuted APL language interpreter and associated utilities. The software in this subset is not supported by DIGITAL.	
ULXBASE400, UDXBASE400	Base Extension Programs and data files that may be useful in some environments. These include obsolete boot programs, drivers for unsupported devices, the `troff` package with fonts, and miscellaneous software. The software in this subset is not supported by DIGITAL.	
ULXBIB400*, UDXBIB400*	Bibliographic Utilities Programs and data useful in maintaining bibliographic information. The software in this subset is not supported by DIGITAL.	Requires: ULTDCMT400, UDTDCMT400
ULXCOURIER400*, UDXCOURIER400*	Remote Procedure Call Compiler Modules for producing software using the COURIER remote procedure call protocol. The software in this subset is not supported by DIGITAL.	
ULXCPM400*	CP/M 8in Diskette Utility Utilities for reading and writing 8-inch diskettes used with the CP/M operating system. The software in this subset is not supported by DIGITAL.	
ULXDOC400, UDXDOC400	Supplementary Documentation Online supplementary documentation set.	
ULXEDIT400, UDXEDIT400	GNU Emacs The Public Domain GNU Emacs editor and edit-macro files. The software in this subset is not supported by DIGITAL.	
ULXF77400	FORTRAN-77 Utilities for developing programs using the UNIX F77 dialect of FORTRAN.	
ULXGAMES400*, UDXGAMES400*	Games and Diversions Programs used for entertainment. The software in this subset is not supported by DIGITAL.	
ULXHYPER400*	Hyperchannel Utilities The Hyperchannel driver and associated daemons and utilities. The software in this subset is not supported by DIGITAL.	
ULXICON400*	ICON (Language) Development Package Translator and linker for the ICON programming language. The software in this subset is not supported by DIGITAL.	

Table 3: (continued)

Subset Name	Contents	Dependencies
ULXINGRES400	University Ingres QUEL DBMS INGRES database management system. The software in this subset is not supported by DIGITAL.	
ULXLEARN400, UDXLEARN400	Computer-Aided System Tutor Software for the `learn` program, which provides lessons in aspects of the computing environment. This subset is useful for persons new to the UNIX environment. The software in this subset is not supported by DIGITAL.	
ULXLISP400	Franz Lisp Development Package Programs that make up the Franz Lisp program development environment, including interpreter, libraries, and compiler. The software in this subset is not supported by DIGITAL.	
ULXMAN400, UDXMAN400	Unsupported On-Line Manuals On-line reference pages for programs found in the ULXBASE400 or UDXBASE400 subset. The software in this subset is not supported by DIGITAL.	
ULXMOD2400	Modula-2 Development Package MODULA-2 compiler and libraries. The software in this subset is not supported by DIGITAL.	
ULXNEWS400, UDXNEWS400	USENET News Interface Software Software needed to participate in the USENET news network. The software in this subset is not supported by DIGITAL.	Requires: ULTCOMM400, ULTUUCP400 or UDTCOMM400, UDTUUCP400
ULXNOTES400, UDXNOTES400	Notesfiles Package Software that lets you establish `notesfiles` on your ULTRIX system. The software in this subset is not supported by DIGITAL.	
ULXRCS400, UDXRCS400	Revision Control System Programs that make up a package similar to the SCCS facility provided with the supported software subsets. The software in this subset is not supported by DIGITAL.	
ULXSHELLS400*, UDXSHELLS400*	Auxiliary Command Line Interpreters The 'distributed' shell and a version of the C-shell that features command completion. The software in this subset is not supported by DIGITAL.	

Table 3: (continued)

Subset Name	Contents	Dependencies
ULXSPMS400	Software Project Management System A package useful for managing large software development efforts. The software in this subset is not supported by DIGITAL.	
ULXTOOLS400*, UDXTOOLS400*	Miscellaneous User-Contributed Utilities Small utilities contributed by the user community. The software in this subset is not supported by DIGITAL.	
ULXVARIAN400	Raster Plotter Package Software used to typeset documents on the VARIAN typesetter. The software in this subset is not supported by DIGITAL.	Requires: ULTDCMT400

* User-contributed software.

Name

efl – Extended FORTRAN Language

Syntax

efl [*option* ...] [*filename* ...]

Description

The `efl` command compiles a program written in the EFL language into clean FORTRAN. The `efl` command provides the same control flow constructs as does `ratfor`(1), which are essentially identical to those in C:

statement grouping with braces;
: decision-making with if, if-else, and switch-case; while, for, FORTRAN do, repeat, and repeat...until loops; multi-level break and next. In addition, EFL has C-like data structures, and more uniform and convenient input/output syntax, generic functions. EFL also provides some syntactic sugar to make programs easier to read and write:

free form input:
: multiple statements/line; automatic continuation statement label names (not just numbers),

comments:
: # this is a comment

translation of relationals:
: >, >=, etc., become .GT., .GE., etc.

return (expression)
: returns expression to caller from function

define:
: define name replacement

include:
: include filename

The `efl` program is best used with `f77`(1).

Options

–w Suppresses warning messages.

–C Causes comments to be copied through to the FORTRAN output (default).

–# Prevents comments from being copied through. If a command argument contains an embedded equal sign, that argument is treated as if it had appeared in an option statement at the beginning of the program.

See Also

f77(1), ratfor(1).
S. I. Feldman, *The Programming Language EFL*, Bell Labs Computing Science Technical Report #78.

eqn (1)
Unsupported

Name

eqn, neqn, checkeq – typeset mathematics

Syntax

eqn [**–d***xy*] [**–p***n*] [**–s***n*] [**–f***n*] [*file*] ...
checkeq [*file*] ...

Description

The `eqn` command is a troff(1) preprocessor for typesetting mathematics on a Graphic Systems phototypesetter, `neqn` on terminals. Usage is almost always

```
eqn file ... | troff

neqn file ... | nroff
```

If no files are specified, these programs reads from the standard input. A line beginning with .EQ marks the start of an equation; the end of an equation is marked by a line beginning with .EN. Neither of these lines is altered, so they may be defined in macro packages to get centering, numbering, etc. It is also possible to set two characters as delimiters; subsequent text between delimiters is also treated as `eqn` input. Delimiters may be set to characters *x* and *y* with the command-line argument **–d***xy* or (more commonly) with 'delim *xy*' between .EQ and .EN. The left and right delimiters may be identical. Delimiters are turned off by delim off. All text that is neither between delimiters nor between .EQ and .EN is passed through untouched.

The program *checkeq* reports missing or unbalanced delimiters and .EQ/.EN pairs.

Tokens within `eqn` are separated by spaces, tabs, newlines, braces, double quotes, tildes or circumflexes. Braces { } are used for grouping; generally speaking, anywhere a single character like *x* could appear, a complicated construction enclosed in braces may be used instead. Tilde ~ represents a full space in the output, circumflex ^ half as much.

Subscripts and superscripts are produced with the keywords **sub** and **sup.** Thus *x sub i* makes x_i, *a sub i sup 2* produces a_i^2, and *e sup {x sup 2 + y sup 2}* gives $e^{x^2+y^2}$.

Fractions are made with **over**: *a over b* yields $\frac{a}{b}$.

sqrt makes square roots: *1 over sqrt {ax sup 2 +bx+c}* results in $\frac{1}{\sqrt{ax^2+bx+c}}$.

The keywords **from** and **to** introduce lower and upper limits on arbitrary things:
$\lim_{n\to\infty} \sum_{0}^{n} x_i$ is made with *lim from {n–> inf } sum from 0 to n x sub i.*

Left and right brackets, braces, and so forth, of the right height are made with **left** and **right:**

left [x sup 2 + y sup 2 over alpha right] ~=~1 produces $\left[x^2+\frac{y^2}{\alpha}\right] = 1.$

The **right** clause is optional. Legal characters after **left** and **right** are braces, brackets, bars, **c** and **f** for ceiling and floor, and "" for nothing at all (useful for a right-side-only bracket).

Vertical piles of things are made with **pile**, **lpile**, **cpile**, and **rpile**: *pile {a above b above c}* produces $\begin{matrix} a \\ b \\ c \end{matrix}$. There can be an arbitrary number of elements in a pile. **lpile** left-justifies, **pile** and **cpile** center, with different vertical spacing, and **rpile** right justifies.

Matrices are made with **matrix**: *matrix { lcol { x sub i above y sub 2 } ccol { 1 above 2 } }* produces $\begin{matrix} x_i & 1 \\ y_2 & 2 \end{matrix}$. In addition, there is **rcol** for a right-justified column.

Diacritical marks are made with **dot**, **dotdot**, **hat**, **tilde**, **bar**, **vec**, **dyad**, and **under**: *x dot = f(t) bar* is $\dot{x}=\overline{f\,(t)}$, *y dotdot bar ~=~ n under* is $\bar{\ddot{y}} = \underline{n}$, and *x vec ~=~ y dyad* is $\vec{x} = \overleftrightarrow{y}$.

Sizes and font can be changed with **size** *n* or **size** ±*n*, **roman**, **italic**, **bold**, and **font** *n*. Size and fonts can be changed globally in a document by **gsize** *n* and **gfont** *n*, or by the command-line arguments **–s***n* and **–f***n*.

Normally subscripts and superscripts are reduced by 3 point sizes from the previous size; this may be changed by the command-line argument **–p***n*.

Successive display arguments can be lined up. Place **mark** before the desired lineup point in the first equation; place **lineup** at the place that is to line up vertically in subsequent equations.

Shorthands may be defined or existing keywords redefined with **define**: *define thing % replacement %* defines a new token called *thing* which will be replaced by *replacement* whenever it appears thereafter. The % may be any character that does not occur in *replacement*.

Keywords like *sum* (∑) *int* (∫) *inf* (∞) and shorthands like >= (≥) -> (→), and != (≠) are recognized. Greek letters are spelled out in the desired case, as in *alpha* or *GAMMA*. Mathematical words like sin, cos, log are made Roman automatically. *Troff*(1) four-character escapes like \(bs (\) can be used anywhere. Strings enclosed in double quotes "..." are passed through untouched; this permits keywords to be entered as text, and can be used to communicate with *troff* when all else fails.

Restrictions

To embolden digits, parens, etc., it is necessary to quote them, as in bold "12.3".

See Also

troff(1), tbl(1), ms(7), eqnchar(7)
ULTRIX Programmer's Manual, Unsupported
B. W. Kernighan and L. L. Cherry, *Typesetting Mathematics—User's Guide*
J. F. Ossanna, *NROFF/TROFF User's Manual*

Name

f77 – Fortran 77 compiler

Syntax

f77 [*option...*] *file...*

Description

The `f77` command runs the ULTRIX Fortran 77 compiler.

Arguments

The `f77` command accepts several types of *file* arguments:

Arguments whose names end with '.f' are taken to be Fortran 77 source programs; they are compiled, and each object program is left on the file in the current directory whose name is that of the source with '.o' substituted for '.f'.

Arguments whose names end with '.F' are also taken to be Fortran 77 source programs; these are first processed by the C preprocessor before being compiled by `f77`.

Arguments whose names end with '.r' or '.e' are taken to be Ratfor or EFL source programs respectively; these are first transformed by the appropriate preprocessor, then compiled by f77.

Arguments whose names end with '.c' or '.s' are taken to be C or assembly source programs and are compiled or assembled, producing a '.o' file.

Other arguments are taken to be F77-compatible object programs, typically produced by an earlier run, or libraries of F77-compatible routines. These programs, together with the results of any compilations specified, are loaded in the order given to produce an executable program with the name ''a.out''.

Programs compiled with `f77` produce memory dumps in file `core` upon abnormal termination if the **–g** option was specified during loading. If the environment variable *f77_dump_flag* is set to a value beginning with y or n, dumps for abnormal terminations are respectively forced or suppressed.

Options

The following options have the same meaning as in `cc`(1). See `ld`(1) for load-time options.

–c Suppresses loading and produces '.o' files for each source file.

–g Produces additional symbol table information for `dbx`(1) and pass the **–lg** flag to `ld`(1) so that on abnormal terminations, the memory image is written to file *core*. Incompatible with **–O**.

–o *output*
Name the final output file *output* instead of `a.out`.

–p Prepare object files for profiling, see `prof`(1).

–pg Causes the compiler to produce counting code in the manner of **–p,** but invokes

a run-time recording mechanism that keeps more extensive statistics and produces a *gmon.out* file at normal termination. An execution profile can then be generated by use of `gprof`(1).

–w Suppresses all warning messages. If the option is '–w66', only Fortran 66 compatibility warnings are suppressed.

–D*name=def*

–D*name*
Defines the *name* to the C preprocessor, as if by '#define'. If no definition is given, the name is defined as "1". ('.F' suffix files only).

–I*dir* '#include' files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in **–I** options, then in directories on a standard list. ('.F' suffix files only).

–O Invoke an object-code optimizer. Incompatible with **–g**.

–S Compiles the named programs, and leave the assembler-language output on corresponding files suffixed '.s'. (No '.o' is created.).

The following options are peculiar to `f77`.

–d Used for debugging the compiler.

–i2 On machines which support short integers, make the default integer constants and variables short. (**–i4** is the standard value of this option). All logical quantities will be short.

–q Suppresses printing of file names and program unit names during compilation.

–m Apply the M4 preprocessor to each '.r' file before transforming it with the Ratfor or EFL preprocessor.

–onetrip

–1 Compile DO loops that are performed at least once if reached. (Fortran 77 DO loops are not performed at all if the upper limit is smaller than the lower limit.)

–r8 Treat all floating point variables, constants, functions and intrinsics as double precision and all complex quantities as double complex.

–u Make the default type of a variable 'undefined' rather than using the default Fortran rules.

–v Print the version number of the compiler, and the name of each pass as it executes.

–C Compile code to check that subscripts are within declared array bounds. For multi-dimensional arrays, only the equivalent linear subscript is checked.

–F Apply the C preprocessor to '.F' files, and the EFL, or Ratfor preprocessors to '.e' and '.r' files, put the result in the file with the suffix changed to '.f', but do not compile.

–E*x* Use the string *x* as an EFL option in processing '.e' files.

–R*x* Use the string *x* as a Ratfor option in processing '.r' files.

–N[qxscn]*nnn*
Make static tables in the compiler bigger. The compiler will complain if it overflows its tables and suggest you apply one or more of these flags. These flags have the following meanings:

- **q** Maximum number of equivalenced variables. Default is 150.
- **x** Maximum number of external names (common block names, subroutine and function names). Default is 200.
- **s** Maximum number of statement numbers. Default is 401.
- **c** Maximum depth of nesting for control statements (e.g. DO loops). Default is 20.
- **n** Maximum number of identifiers. Default is 1009.

–U Do not convert upper case letters to lower case. The default is to convert Fortran programs to lower case except within character string constants.

Restrictions

Files longer than about 50,000 lines must be split up to be compiled.

Diagnostics

The diagnostics produced by `f77` itself are intended to be self-explanatory. Occasional messages may be produced by the loader.

Files

file.[fFresc]	Input file
file.o	Object file
a.out	Loaded output
/usr/lib/f77pass1	Compiler
/lib/f1	Pass 2
/lib/c2	Optional optimizer
/lib/cpp	C preprocessor
/usr/lib/libF77.a	Intrinsic function library
/usr/lib/libI77.a	Fortran I/O library
/usr/lib/libU77.a	ULTRIX interface library
/usr/lib/libm.a	math library
/lib/libc.a	C library, see section 3
/usr/lib/libF77_p.a	Profiling intrinsic function library
/usr/lib/libI77_p.a	Profiling Fortran I/O library
/usr/lib/libU77_p.a	Profiling ULTRIX interface library
/usr/lib/libm_p.a	Profiling math library
/usr/lib/libc_p.a	Profiling C library, see section 3
mon.out	File produced for analysis by prof(1)
gmon.out	File produced for analysis by gprof(1)

See Also

ar(1), cc(1), dbx(1), efl(1), fpr(1), fsplit(1), gprof(1), ld(1), prof(1), ranlib(1), ratfor(1), struct(1), intro(3f)
"Introduction to the f77 I/O Library", and "A Portable Fortran 77 Compiler,"
ULTRIX Supplementary Documents, Vol. II:Programmer

Name

fed – font editor

Syntax

fed [**–i**] [**–q**] name

Description

The `fed` program is an editor for font files. It is display oriented and must be used on an HP 2648 graphics terminal. `fed` does the necessary handshaking to work at 9600 baud on the 2648.

Options

–i Requests *inverse video mode*, where all dots are dark and the background is bright. This provides a setting similar to the hardcopy output of the plotter, and is useful for fonts such as the shadow font where shading is important.

–q Requests *quiet mode*, where all graphic output is suppressed. This mode is useful on terminals other than the HP 2648 (assuming you are editing blindly) and for operations such as the # and A commands, since these operations do not make essential use of graphics, and since suppression of the graphic output speeds of `fed` considerably.

Restrictions

Attempting to use the second 128 characters would be folly. Fed has never been tested on such fonts, and at a bare minimum there would be problems trying to input 8 bit characters.

The character DEL is interpreted by the tty driver to mean interrupt. Hence the corresponding glyph cannot be accessed. The *start*, *stop*, and *quit* characters are turned off, but other characters used by the new tty driver must be quoted with ^V.

Changed widths are not copied to the width table used by troff. This only matters if logical widths are changed, or if glyphs are moved around. For these cases, *vwidth(1)* must be used.

Fonts

A font is a collection of up to 256 *glyphs*, each of which is some pattern or design. Glyphs are represented on Unix as a rectangular array of dots, each of which is either dark or blank. Each location in the array is called a *pixel*. There are 200 pixels per inch due to the hardware of the Versatec and Varian plotters.

Each glyph has, in addition to its bit pattern, a *base* and a *width*. The base is a point, typically near the lower left of the array, that represents the logical lower left point of the glyph. The base is not restricted to be within the array, in fact, it is usually a few locations to the left of the edge. The vertical position of the base defines the *baseline*, which is held constant for all glyphs when a line is typeset. Letters with descenders, such as "g", go below the baseline. Other glyphs typically rest on the baseline.

The width is used by *troff(1)* to determine where to place the next glyph. It need not be the same as the width of the array, although it is usually about the same.

The size of the array, location of the base, and the width can vary among glyphs in a font. Fonts where all glyphs have the same width are called *fixed width fonts*, others are *variable width fonts*.

Attributes which do not vary among glyphs include the *font name*, which can be up to 11 alphabetic characters, and the *point size*, which is a positive integer indicating the overall size of the font. A point is 1/72 inch. The point size of a font is the distance, in points, from the top of the tallest glyph to the bottom of the lowest. The software of troff currently restricts point sizes to 6, 7, 8, 9, 10, 11, 12, 14, 16, 18, 20, 22, 24, 28, and 36 point. Normal text is usually 10 point.

Font files conventionally have names of the form

name.pointsize

for example, 'bocklin.14' to indicate 14 point bocklin. Fed will look for such a file in both the current directory and /usr/lib/vfont. Vtroff will only look in /usr/lib/vfont.

There is a correspondence between *glyphs* and *characters* in a font. For a given font, each glyph has an ASCII character associated with it. The glyph is obtained in troff by typing the associated character, and in fed glyphs are also referred to by their character. However, it is not required for all characters to have a glyph, fonts never have more than 128 glyphs and usually have fewer.

There is usually a natural correspondence between glyphs and characters. For example, the glyph which is a roman lower case 'a' will generally have the ascii character 'a' as its corresponding character. In the special font, the Greek lower case alpha has 'a' as it's corresponding character, upper case delta has 'D' as it's corresponding character, etc. However, special fonts such as the chess font have glyphs that do not appear to be related to their corresponding characters.

It is easy to confuse glyphs and characters. Note, however, that the three glyphs roman a, bold **a**, and italic *a*, are all different, yet all three correspond to the character 'a'. When this is multiplied by the large number of font styles and point sizes, there are many glyphs that match a single character, (but only one in a particular font).

Fed Organization

Fed organizes the screen into 21 *windows* in a 3 by 7 array. Each window is 100 by 100 pixels, meaning that the maximum height and width of a glyph is 100 pixels. Since the HP 2648 has a resolution of 100 dots per inch, glyphs displayed on the screen and printer will be double the actual height and width, even when fully zoomed out. There is a *current window*, which will be marked with a square border. There are two *pens*, called *fine* and *bold*. The fine pen is one pixel wide, the bold pen can range from two pixels to ten pixels in diameter. The default width of the bold pen is taken from the point size implied by the file name. The point size is not otherwise used. There are also fine and bold *erasers*.

There are two locations in the window, called the *cursor* and the *mark*. These tools are used to draw on glyphs.

Sometimes the cursor is on, in which case it is indicated by the hardware graphics cursor of the terminal, a cross. The cursor is considered to be located at the center of the cross. Sometimes the *rubber band line* is turned on, showing the path a line

drawn would traverse. This line runs from the mark to the cursor, and is the only way the mark is graphically visible.

Commands

Commands to fed are single characters, sometimes followed by any needed arguments. The commands used by fed were chosen to be as similar to *vi*(1) commands as was reasonable. Another distinction is that certain commands are in upper case. These commands were deliberately made hard to type because they cause a large change in the state of the editor and should not be done by accident. In a few cases there are both upper and lower case commands with the same letter.

Alphanumeric Keypad: Note that this is the keypad on the far right. The graphics keypad on the near right will not work. These keys are each synonyms for other commands. They are arranged in a manner that causes the five arrow keys to behave sensibly, but the others need to be memorized or stickers placed on the keys. They are provided for convenience only, and the user can avoid memorization simply by using the mnemonic letter keys instead.

The layout is as follows:

undo (u)	rezoom ( )	fillin (f)
move (m)	up (k)	draw (d)
left (h)	base (b)	right (l)
setdot (.)	down (j)	cleardot (>)

The arrow keys move the cursor one pixel in the indicated direction. The cursor is turned on if it was off. Note that the alphanumeric keys (far right) must be used. The graphics keys (near right) will appear to move the cursor but it will not be moved internally. The cursor cannot be moved outside the current window.

^*L*: Redraw the screen. This is useful if an I/O error or background process has caused the screen to get messed up.

b: Move the cursor to the base of the window. This is the default location of the cursor.

c: If the cursor is on, turn it off. Otherwise, turn it on.

d: Draw a line from the mark to the cursor. The currently selected tool (fine pen, bold pen, fine eraser, bold eraser) is used. The cursor is turned off. The mark is moved to the location of the cursor.

f: Fill in the current hole. The cursor must be in a completely enclosed empty (white) area. The area is set to black. If this command is invoked on the outside or there are any leaks to the outside, the entire outside will be filled in. (Undo is useful in this case.) Filling in cannot jump diagonals, but can rather only spread in the four orthogonal directions.

g <x>: Get a glyph. X can be any character. The glyph corresponding to x is put in a window, and this window is made the current window. The glyph is centered horizontally in the window. The baseline is located at row 70 from the top of the window. The pen and cursor are placed at the base, and the cursor is turned off. The glyph must exist.

h, *j*, *k*, and *l* are accepted to mean left, down, up, and right, respectively. They are synonymous with the alphanumeric arrow keys. They have the same meanings as in *vi(1)*.

m: Move the mark to the current location of the cursor. The cursor is turned on.

n <x>: New glyph. This is similar to *g*, except that the glyph must *not* exist. It is used to create a new glyph. A blank window is created, centered at (50, 70) as in g.

p: Print the contents of the screen. An HP 2631 printer must be connected to the terminal. The screen is copied to the printer. If in inverse video mode, the screen is changed to normal video mode before the print, and then changed back after the print.

r: If the rubber band line is on, turn it off. Otherwise, turn it on.

s <what> [<where>]: Set <what> to <where>. What and where are single characters. The possibilities are:

spf: Set pen fine. ('l' for light is also accepted.)

spb: set pen bold. ('h' for heavy is also accepted.)

sd: Set draw. The pen is used instead of the eraser.

se: Set erase. The eraser is used instead of the pen.

ss<n>: Set size of bold pen. <n> is a digit from 1 to 9. The size of the bold pen is set accordingly. This also affects the bold eraser.

u: Undo. The previous change to the current window is undone. Note that undo is on a window by window basis, so that commands that affect characters or more than one window cannot be undone.

z <n>: Zoom to level n. The screen is blown up by a factor of n. This only affects the appearance of the screen to make it easy to see the individual dots, and does not affect the size of the glyph or the result of a print command. Zooming to 1 shows the entire screen, a level of 3 or 4 is probably good for editing glyphs. When a message is printed on the screen, fed automatically zooms out to level 1 so you can read the message. Hitting space will zoom back. z followed by <return> zooms out without changing the previous zoom.

space: Zoom back to the level most recently requested by the z command.

A <i/e/r> <first> <last> [<oldps> <newps>]:
Artificially italicize/embolden/resize a range of glyphs in the current font. Enter i for italicize, e for embolden, or r for resize, and the first and last character in the range desired. If you are resizing you will also have to enter the old and new point size, each terminated by a return. Each glyph is gotten and changed on the screen visibly. Glyphs are italicized by slanting them to the right at a slope of 1/5. They are emboldened by smearing them to the right a number if pixels equal to the current heavy pen size. They are resized with an algorithm which translates all on bits to the new position. These operations will be considerably faster if the **–q** option is in effect, since much overhead is involved in the graphic display.

B: Move the base to the cursor. The cursor is turned on.

C <from> <to>: Copy the glyph in character <from> to character <to>. If <from> has a window on the screen, that window is given to <to>.

D <from> <through>: Delete a range of characters in the font, from <from> through <through> inclusive. To delete a single character type it twice.

E <file>: Edit the named file. If changes have been made to the current file, confirmation will be requested. (Either 'y' or 'E' is accepted.) The file name is terminated with return.

F <first> <last>: Show the font on the screen. The characters in the specified range are shown. The width values are used to get natural spacing. The display will remain until another command is typed, at which time the previous display will be redrawn and the new command will be executed. As a special case, a ''p'' command will print the results of the ''F'' command instead of the previous display.

I <h/v>: Invert the current glyph about a horizontal or vertical axis, as indicated by *h* or *v*. The axis runs up the center of the window. The base can be subsequently positioned with the *B* command.

K: Kill the current glyph. All dots are set to blank. The glyph is not removed from the font. This is used for redrawing a glyph from scratch or replacing it with another glyph.

M <from> <to>: Move a glyph from <from> to <to>. This is just like the copy command but the original is deleted.

N <file>: Write out the current file, if necessary, and edit the new file specified. The file name is terminated with return.

P <first> <last> <file>: Partial read from a file. A file and the first and last characters in the range are prompted for. Characters not in the range are left unmodified, characters in the range are handled as in the R command.

Q: Quit the editor, without saving any work. If changes have been made confirmation will be required (either 'Q' or 'y' is taken as 'yes'.)

R <file>: Read in the named file on top of the current file. Glyphs are merged wherever possible. If there is a conflict, you will be asked whether fed should take the glyph from the file (f) or buffer (b). Responding with F or B will lock in that mode for the remainder of the read. The file name is terminated with a return.

T <text>: Typeset the line of text on the terminal. This is similar to the F command except that the given text is arranged on the screen, so you can see how some particular combination of characters would look.

V: Toggle whether editing is being done in inverse video mode.

W <file>: Write the buffer out onto the named file, which is terminated by return. A null file name means the current file name.

ZZ: Exit fed. A write is done, if necessary, followed by a quit. This is the normal way to leave fed. The Z must be doubled for compatibility with *vi*.

.: Turn on the dot under the cursor. The cursor is turned off.

>: Turn off the dot under the cursor. The cursor is turned off.

 # <char> <field> <value>: Edit a numerical field. This only makes sense if the glyph has not been gotten (*g* or *n*) yet, since otherwise the values are taken from window specific things such as the base. Fed does not do any sanity checking, but just substitutes the value input. Fields are the first letter of any field from the dispatch structure (see vfont(5)), specifically, these fields are *addr*, *nbytes*, *left*, *right*, *up*, *down*, and *width*. The number, which may be signed, is terminated by a newline.

Files

/usr/lib/vfont/*.*

See Also

vfont(5), vfontinfo(1), vtroff(1), vwidth(1)

fp(1)
Unsupported

Name

fp – Functional Programming language compiler/interpreter

Syntax

fp

Description

The `fp` interpreter/compiler implements the applicative language proposed by John Backus. It is written in FRANZ LISP .

In a functional programming language intent is expressed in a mathematical style devoid of assignment statements and variables. Functions compute by value only; there are no side-effects since the result of a computation depends solely on the inputs.

The `fp` programs consist of *functional expressions* – primitive and user-defined `fp` functions combined by *functional forms*. These forms take functional arguments and return functional results. For example, the composition operator '@' takes two functional arguments and returns a function which represents their composition.

There exists a single operation in `fp` – *application*. This operation causes the system to evaluate the indicated function using the single argument as input (all functions are monadic).

Getting Started

`fp` invokes the system. `fp` compiles functions into *lisp*(1) source code; *lisp*(1) interprets this code (the user may compile this code using the *liszt*(1) compiler to gain a factor of 10 in performance). Ctrl D exits back to the shell. *Break* terminates any computation in progress and resets any open file units. *)help* provides a short summary of all user commands.

Restrictions

If a non-terminating function is applied as the result of loading a file, then control is returned to the user immediately, everything after that position in the file is ignored.

`fp` incorrectly marks the location of a syntax error on large, multi-line function definitions or applications. (Turing award lecture by John Backus).

Files

/usr/ucb/lisp	the FRANZ LISP interpreter
/usr/ucb/liszt	the liszt compiler
/usr/doc/fp	the User's Guide

See Also

lisp(1), liszt (1).
The Berkeley FP user's manual, available on-line. The language is described in the August 1978 issue of *CACM*

Name

fpr – print FORTRAN file

Syntax

fpr

Description

The `fpr` filter transforms files formatted according to FORTRAN's carriage control conventions into files formatted according to UNIX line printer conventions.

The `fpr` filter copies its input onto its output, replacing the carriage control characters with characters that will produce the intended effects when printed using `lpr`(1). The first character of each line determines the vertical spacing as follows:

Character	Vertical Space Before Printing
Blank	One line
0	Two lines
1	To first line of next page
+	No advance

A blank line is treated as if its first character is a blank. A blank that appears as a carriage control character is deleted. A zero is changed to a newline. A one is changed to a form feed. The effects of a "+" are simulated using backspaces.

Restrictions

Results are undefined for input lines longer than 170 characters.

Examples

```
% a.out | fpr | lpr -Pprinter
% fpr < f77.output | lpr -Pprinter
```

Name

help – tips on getting started with an ULTRIX system

Syntax

help

Description

The `help` command displays information on how to get started using an ULTRIX operating system. It recommends places to start with the ULTRIX documentation, and gives a list of commands helpful to beginners.

Name

learn – computer aided instruction about UNIX

Syntax

learn [*–directory*] [*subject* [*lesson*]]

Description

The `learn` command gives computer aided instruction courses and practice in the use of UNIX, the C Shell, and the Berkeley text editors. To get started simply type **learn**. The program will ask questions to find out what you want to do. Some questions may be bypassed by naming a *subject*, and more yet by naming a *lesson*. You may enter the *lesson* as a number that `learn` gave you in a previous session. If you do not know the lesson number, you may enter the *lesson* as a word, and `learn` will look for the first lesson containing it. If the *lesson* is '–', `learn` prompts for each lesson; this is useful for debugging.

The *subject*s presently handled are

files
editor
vi
morefiles
macros
eqn
C

There are a few special commands. The command 'bye' terminates a `learn` session and 'where' tells you of your progress, with 'where m' telling you more. The command 'again' re-displays the text of the lesson and 'again *lesson*' lets you review *lesson*.

Options

–directory Allows one to exercise a script in a nonstandard place.

Restrictions

The main strength of `learn`, that it asks the student to use the real UNIX, also makes possible baffling mistakes. It is helpful, especially for nonprogrammers, to have a UNIX initiate near at hand during the first sessions.

Occasionally lessons are incorrect, sometimes because the local version of a command operates in a non-standard way. Such lessons may be skipped with the 'skip' command, but it takes some sophistication to recognize the situation.

To find a *lesson* given as a word, `learn` does a simple `fgrep`(1) through the lessons. It is unclear whether this sort of subject indexing is better than none.

Spawning a new shell is required for each of many user and internal functions.

Files

/usr/lib/learn	subtree for all dependent directories and files
/usr/tmp/pl*	playpen directories

See Also

csh(1), ex(1)

Name

lisp – lisp interpreter

Syntax

lisp

Description

The `lisp` interpreter is for a dialect which closely resembles MIT's MACLISP. This lisp, known as FRANZ LISP, features an I/O facility which allows the user to change the input and output syntax, add macro characters, and maintain compatibility with upper-case only lisp systems; infinite precision integer arithmetic, and an error facility which allows the user to trap system errors in many different ways. Interpreted functions may be mixed with code compiled by *liszt*(1) and both may be debugged using the "Joseph Lister" trace package. A `lisp` containing compiled and interpreted code may be dumped into a file for later use.

There are too many functions to list here; one should refer to the manuals listed below.

Files

/usr/lib/lisp/trace.l	Joseph Lister trace package
/usr/lib/lisp/toplevel.l	top level read-eval-print loop

See Also

liszt(1), lxref(1)
'FRANZ LISP Manual, Version 1' by John K. Foderaro
MACLISP Manual

Name

liszt – compile a Franz Lisp program

Syntax

liszt [**–mpqruwxCQST**] [**–e** *form*] [**–o** *objfile*] [*name*]

Description

The `liszt` compiler takes a file whose names ends in '.l' and compiles the FRANZ LISP code there leaving an object program on the file whose name is that of the source with '.o' substituted for '.l'.

Options

–e Evaluate the given form before compilation begins.

–m Compile a MACLISP file, by changing the readtable to conform to MACLISP syntax and including a macro-defined compatibility package.

–o Put the object code in the specified file, rather than the default '.o' file.

–p Places profiling code at the beginning of each non-local function. If the lisp system is also created with profiling in it, this allows function calling frequency to be determined (see *prof*(1).)

–q Only print warning and error messages. Compilation statistics and notes on correct but unusual constructs will not be printed.

–r Place bootstrap code at the beginning of the object file, which when the object file is executed will cause a lisp system to be invoked and the object file fasl'ed in.

–u Compile a UCI-lispfile, by changing the readtable to conform to UCI-Lisp syntax and including a macro-defined compatibility package.

–w Suppress warning diagnostics.

–x Create a lisp cross reference file with the same name as the source file but with '.x' appended. The program *lxref*(1) reads this file and creates a human readable cross reference listing.

–C Put comments in the assembler output of the compiler. Useful for debugging the compiler.

-Q Print compilation statistics and warn of strange constructs. This is the default.

–S Compile the named program and leave the assembler-language output on the corresponding file suffixed '.s'. This will also prevent the assembler language file from being assembled.

–T Send the assembler output to standard output.

If no source file is specified, then the compiler will run interactively. You will find yourself talking to the *lisp*(1) top-level command interpreter. You can compile a file by using the function *liszt* (an nlambda) with the same arguments as you use on the command line. For example to compile 'foo', a MACLISP file, you would use:

```
(liszt –m foo)
```

Note that *liszt* supplies the ''.l'' extension for you.

Files

/usr/lib/lisp/machacks.l	MACLISP compatibility package
/usr/lib/lisp/syscall.l	macro definitions of Unix system calls
/usr/lib/lisp/ucifnc.l	UCI Lisp compatibility package

See Also

lisp(1), lxref(1)

VAX **lxref(1)**
Unsupported

Name

lxref – lisp cross reference program

Syntax

lxref [**–N**] *xref-file* ... [**–a** *source-file* ...]

Description

The `lxref` command reads cross reference file(s) written by the lisp compiler `liszt` and prints a cross reference listing on the standard output. `liszt` will create a cross reference file during compilation when it is given the **–x** switch. Cross reference files usually end in '.x' and consequently `lxref` will append a '.x' to the file names given if necessary. The first option to `lxref` is a decimal integer, N, which sets the *ignorelevel*. If a function is called more than *ignorelevel* times, the cross reference listing will just print the number of calls instead of listing each one of them. The default for *ignorelevel* is 50.

The **–a** option causes `lxref` to put limited cross reference information in the sources named. `lxref` will scan the source and when it comes across a definition of a function (that is a line beginning with '(def' it will precede that line with a list of the functions which call this function, written as a comment preceded by **';.. '** . All existing lines beginning with **';.. '** will be removed from the file. If the source file contains a line beginning **';.-'** then this will disable this annotation process from this point on until a **';.+'** is seen (however, lines beginning with **';.. '** will continue to be deleted). After the annotation is done, the original file named 'foo.l' is renamed to '#.foo.l' and the new file with annotation is named 'foo.l'

See Also

lisp(1), liszt(1)

Name

mod – Modula-2 compiler

Syntax

mod [*options*] *name* ...

Description

The `mod` command compiles one or more Modula-2 programs or implementation modules. Definition modules are not compiled. In the absence of options, it will compile all specified modules and link them together into an executable file called `a.out`.

Each program or implementation module must be in a separate file with a name ending with .mod. Each definition module must be in a separate file called module.def where module is the name of the module. Object files ending with .o compiled with `mod` or some other compiler may be specified.

File name arguments ending with .pcd, and .s, are assumed to be –pcode and assembly language files respectively, and are translated and assembled into object files.

Options

–c Create object files but do not link them together.

–g Generate additional symbol table information for the debugger `dbx`(1).

–i Ignore the errors in some of the modules and continue compiling the rest of them.

–m flags Perform intermodule checking. If an out-of-date module is encountered, recompile it using the specified *flags*. The flags are separated by commas or spaces, and must be quoted if spaces are used.

–n Write out what will happen when the same command is entered without the ''-n'' option.

–o *name* Create an executable file called ''name'' instead of the default ''a.out''.

–pg Set up object files for profiling by `gprof`(1).

–r Retain pcode and assembly language files in the current directory after compilation.

–s Use standard conventions for reserved word case, cardinal data type, and strings. See Extensions below.

–sc Use standard conventions for cardinal data type (See Extensions, below).

–sk Use standard conventions for reserved word case (See Extensions, below).

–ss Use standard conventions for string constants (See Extensions, below).

–u Convert all identifiers and reserved words to upper case (that is, ignore the case of identifiers and reserved words on input).

–v Print out messages which state what is occurring during compilation.

–C Generate runtime checks for illegal pointers, subrange and index bounds, and variant record tags.

–D *directory*
Use the specified directory for the phases of the compiler and the location of the standard definition modules and libraries.

–L Ignore references to modules not specified while performing intermodule checking. This is useful when checking modules to be placed in a library.

–M Perform intermodule checking, but do not recompile if inconsistencies are found.

–N Ignore references to the module name while performing intermodule checking. This is useful when the module name is not a Modula-2 module. You may use this option as many times as needed.

–O Perform code optimizations.

–P Stop after generating pcode in a file ending with .pcd.

–S Stop after generating assembly language in a file ending with .s.

Library Modules

By default, an import of a global module will cause the compiler to look for the definition module first in the working directory and then in the standard library directory. The standard library modules are automatically linked with the program.

The default may be overridden to specify other directories of definition modules using the MODPATH environment variable. MODPATH is set to a sequence of directory names separated by colons. Those directories will be searched in the order specified to find a definition module. The corresponding object files or libraries are specified when linking. The MODPATH environment variable may be set by the user in .login or in .modpath in the working directory. If the file .modpath exists in the working directory, the `mod` command will use its first line as the value of the MODPATH variable.

The following modules are provided by this implementation of Modula-2. Note that system, memory, io, and bitoperations are builtin modules; definition modules for them are provided for documentation purposes only. Only strings and parameters are actually implemented in Modula-2.

system Built in system module. Contains types of what word, address, etc., and process routines.

memory Built in storage module. Sets up pointers properly for runtime checks. Contains ALLOCATE and DEALLOCATE.

io Built in I/O module that provides formatted read and write similar to `scanf`(3) and `printf`(3).

bitoperations Built in bit manipulation module. Performs operations such as shift, exclusive or, etc., on integer operands.

math Performs mathematical functions. Interface to the C math library.

parameters Accesses command line parameters and environment variables.

strings	Compares, assigns, and links strings.
unix	Defines some UNIX system calls and C library routines.
Storage	Standard storage module, for compatibility with standard Modula-2. Contains ALLOCATE and DEALLOCATE.

Differences And Extensions

This implementation of Modula-2 has compiled and run Wirth's Modula-2 compiler (as modified by Cambridge University for the VAX) with only minor changes to make Wirth's compiler more portable. However, the definition of the language has been relaxed in some areas. For the most part, these changes are compatible.

The following is an incomplete list of differences between this compiler and Wirth's compiler:

Reserved words and standard identifiers are recognized in upper and lower case, Thus, case variations of reserved words may not be used for identifiers. This feature is disabled by the **–sk** option.

Cardinal and non-negative subranges that do not exceed MAXINT are considered to be subranges of integer and are compatible with integers. Subranges that exceed MAXINT are compatible with cardinal and non-negative subranges. This feature is disabled by the **–sc** option.

A built in module called *io* provides formatted input and output. The *readf* and *writef* routines can accept any number of parameters, as long as their types correspond properly with the format string. Supported formats include: for integer and cardinal, **d**, **x**, and **o**; for real, **g** (output only), **f**, and **e**; for longreal, **G** (output only), **F**, and **E**; for char, **c**; and for string (array of char), **s** and [] (input only).

No import of *allocate* or *deallocate* is required to use new and dispose if the standard memory allocation routines are desired. Programs that require checking import *allocate* and *deallocate* from memory rather than storage.

The sizes returned by *size* and *tsize* and expected by *allocate*, *deallocate* and *newprocess* are in units of bits.

The *system* module includes the type *byte*, which is analogous to *word*, as well as appropriate related constants. There is also a function cputime, which returns the accumulated program CPU time in milliseconds.

There is a standard type called *longreal* that stores a double precision real value. A standard function longfloat converts cardinals, integers, or reals to longreal.

Additional standard procedures include:

min(a,b)	Returns the smaller of two cardinal, integer, real, or longreal values.
max(a,b)	Returns the larger of two cardinal, integer, real, or longreal values.
assert(condition[,message])	Aborts the program (with the optional message) if the condition is false.
number(a)	Returns the number of elements in the specified array.
first(type)	Returns the smallest legal value of the specified type.

last(type) Returns the largest legal value of the specified type.

Definition modules are not compiled.

Escape sequences may be placed in strings to specify non-printing characters. E.g., \n, \t, \r, \f, \b, \\, \', and \" mean linefeed, tab, carriage return, form feed, backspace, backslash, single quote, and double quote, respectively. In addition a \ followed by up to three octal digits specifies the the character whose ASCII code is the octal value. A single (double) quote also may be put in a string delimited with single (double) quotes by specifying two single (double) quotes. This feature is disabled by the –ss option.

The interface to Unix is through a module called *unix* rather than the *system* module. The *unixcall* procedure is handled for compatibility with the Cambridge compiler, but is not recommended.

Additional keywords are recognized in certain contexts. These keywords are prefixed by @ to avoid conflicting with valid identifiers.

Pointer attributes

Attributes may be specified between the keywords *pointer* and *to* in order to change the default assumptions of Modula-2 pointer with checking. Recognized attributes are:

@nocheck	Modula-2 pointer, no checking
@c	C/malloc pointer, no checking
@pascal	Pascal pointer, Pascal checking

Size and alignment

The size and alignment of data types may be specified preceding any type specification. The size and alignment multiples are in bits. For example,

```
type Register = @align 2 @size 4 [-8..7];
```

defines a type that occupies 4 bits aligned on a multiple of two bits.

Exports Exports from a definition module are assumed qualified. Unqualified exports are permitted if the @unqualified keyword is used. Multiple export statements are permitted, but they must occur next to each other.

External variables and procedures

A procedure or variable may be accessed by C and Pascal routines using its unqualified name if the @external attribute occurs between the keyword procedure and the name of the procedure or precedes the variable declaration.

Uncounted open arrays

Open array parameters appear as two parameters, the address of the array and the number of element-to-non-Modula-2 programs. If necessary, the count may be omitted by placing the attribute @nocount between the keywords *array* and *of* in the open array declaration.

Restrictions

This is an experimental compiler, and thus no warranties are expressed or implied about its conformance to the definition of the Modula-2 language or about its proper functioning. We will endeavor to report and fix bugs, but users should be aware that this compiler is not a supported product.

Diagnostics

All error messages suppress subsequent compilation phases. Error messages ending with a question mark are internal errors, and probably represent compiler bugs. When pointer checking is running in a Modula-2 program, segmentation faults may be generated by the pointer validation test. These are intentional and should be considered as invalid pointer messages. The compiler runs with runtime checks enabled, and may produce core dumps. Report problems to the author.

Files

file.mod	Program or implementation module
file.def	Definition module
file.pcd	Pcode (–P or –r)
file.s	Assembly code (–S or –r)
/usr/local/lib/mod/mod2.0	Modula-2 compiler front-end
/usr/local/lib/mod/mod2.1	Modula-2 compiler back-end
/usr/local/lib/mod/mod2.2	Intermodule checker
/usr/local/lib/mod/*.def	Standard definition modules
/usr/local/lib/mod/modlib	Default library
/tmp/modNNNNNN.pcd	Temporary Pcode file
/tmp/modNNNNNN.s	Temporary assembly code file

msgs(1)
Unsupported

Name

msgs – system messages and junk mail program

Syntax

msgs [**–fhlpq**] [*number*] [*–number*]
msgs –s
msgs –c [*–days*]

Description

The `msgs` command is used to read system messages. These messages are sent by mailing to the login "msgs" and should be short pieces of information that are suitable to be read once by most users of the system.

The `msgs` command is normally invoked each time you log in, by placing it in the file `.login` (`.profile` if you use `/bin/sh`). It will then prompt you with the source and subject of each new message. If there is no subject line, the first few non-blank lines of the message will be displayed. If there is more to the message, you will be told how long it is and asked whether you wish to see the rest of the message. The possible responses are:

y	Type the rest of the message
RETURN	Synonym for y.
n	Skip this message and go on to the next message.
–	Redisplay the last message.
q	Drops you out of *msgs;* the next time you run the program it will pick up where you left off.
s	Append the current message to the file `Messages` in the current directory; **s–** saves the previously displayed message. Entering **s** or **s–** followed by a space and a filename specifies writing the message to a file other than the default, `Messages`.
m or **m–**	Causes a copy of the specified message to be placed in a temporary mailbox and `mail` to be invoked on that mailbox.

Both **m** and **s** accept a numeric argument in place of the '–'.

The `msgs` command keeps track of the next message you will see by a number in the file `.msgsrc` in your home directory. In the directory `/usr/msgs`, it keeps a set of files whose names are the (sequential) numbers of the messages they represent. The file `/usr/msgs/bounds` shows the low and high number of the messages in the directory so that `msgs` can quickly determine if there are no messages for you. If the contents of `bounds` is incorrect it can be fixed by removing it; `msgs` will make a new `bounds` file the next time you run `msgs`.

Options

–f Causes it not to say "No new messages.". This is useful in your `.login` file since this is often the case here.

–q Queries whether there are messages, printing "There are new messages." if there are. The command "msgs –q" is often used in login scripts.

–h Causes `msgs` to print the first part of messages only.

–l Causes only locally originated messages to be reported.

number causes `msgs` to start at the message specified by *number,* rather than at the next message indicated by your *.msgsrc* file. Thus

% **msgs 5**

causes `msgs` to begin at the fifth message.

–number Causes `msgs` to start *number* messages back from the one indicated by your *.msgsrc* file. This option is useful for reviewing recent messages.

–p Causes long messages to be piped through `more`.

–s Adds a new message to `/usr/msgs`.

–c Removes messages that have been in `/usr/msgs` more than a specified number of days from that file.

–days Determines the number of days a message is in `/usr/msgs` before it is removed. You must be the superuser to use the **–c** option.

Within `msgs` you can also go to any specific message by typing its number when `msgs` requests input.

Files

`/usr/msgs/*`	database
`~/.msgsrc`	number of next message to be presented

See Also

mail(1), more(1)

pti(1)
Unsupported

Name

pti – phototypesetter interpreter

Syntax

pti [*file* ...]

Description

The `pti` command shows the commands in a stream from the standard output of `troff`(1) using `troff`'s **–t** option, interpreting them as they would act on the typesetter. Horizontal motions shows as counts in internal units and are marked with '<' and '>' indicating left and right motion. Vertical space is called lead and is also indicated.

See Also

troff(1)

Name

ptoc – Pascal to C language translator

Syntax

ptoc [*filename*]

Description

The `ptoc` (P to C) translator accepts as input a Pascal source file and produces on the standard output a C language translation. The Pascal input file must not contain any Pascal syntax errors. The `ptoc` translator is not meant to be used as a Pascal debugging tool or as a Pascal syntax checker. The `ptoc` translator does check for some common Pascal syntax errors. If it finds an error, it prints an error message and quits.

The `ptoc` translator accepts standard Pascal (as defined by the ANSI/IEEE770X3.97-1983 standard) with some extensions and restrictions detailed below. It outputs C code that conforms to the portable C compiler, except as noted in the RESTRICTIONS section. The `ptoc` translator converts all identifiers to lower case. This is to conform with Pascal, which is not case sensitive.

The `ptoc` translator will read header files, which are referenced by an include statement, to resolve symbol names that are used in the Pascal file being translated. All Pascal header files which are part of the program being translated must be run through *ptoc* separately, to produce C format header files.

A word about Pascal variant records is in order. The C equivalent of a Pascal variant record is a union. A C union is implemented as a structure with a name, whereas a Pascal variant record is not named, nor does it introduce another structure level. Furthermore, if two or more fields are grouped together in a Pascal variant, they do not require a nested record. In C, however, they must be grouped within a structure. These considerations add one or two extra reference levels in the C code compared to the reference levels needed in the Pascal code.

A common practice in C is to use ''#defines'' to make these reference levels transparent in the code. This technique is used by `ptoc`. Thus, the extra structure levels required by C are transparent in the C code translation of the Pascal source code. For example:

```
Pascal
------
q: record
    i: integer;
    case boolean of
    true: (qa: char;
           qb: integer);
    false: (qf: boolean);
    end;
```

```
C
-
struct q {
    int i;
    union {
        struct {
            char uqa;
            int uqb;
            } un1;
        struct {
            char uqf;
            } un2;
        } un;
    };
#define qa un.un1.uqa
#define qb un.un1.uqb
#define qf un.un2.uqf
```

If a Pascal program has a variable with the same name as one of the field names within a variant record, then the define statement in the C translation will be applied to the variable as well as to the field within the variant. One solution is to change the name of the Pascal variable before running `ptoc`. If you do not change the conflicting variable name, when you try to compile the C translation you may see strange errors such as ''warning: old-fashioned initialization'' or ''warning: illegal member use''.

Extensions

The `ptoc` translator handles a few extensions to standard Pascal which are features of ''popular'' versions of Pascal. These extensions are detailed below.

Reserved Words

The `ptoc` translator accepts upper or lower case Pascal reserved words.

Compilation Units

The `ptoc` translator accepts the keyword ''module'' in place of the keyword ''program''. This is used at the top of a file of separately compiled procedures and functions.

Include Files

Two formats of include files are accepted:

```
Berkeley Pascal                  Alternate
---------------                  ---------
#include "filename"              %include 'filename'
```

Declarations

The ''_'' (underscore) character in symbol names is accepted syntax. Any underscores in symbol names will be kept in the C translation.

The *''"characterinsymbolnamesisacceptedsyntax.Symbolnamesmayalsostartwiththe$character.All''* characters in symbol names will simply be discarded and not output in the C translation.

Arrays of type ''varying'' are accepted and translated into character arrays in C. For example:

```
Pascal                           C
------                           -
varid: varying[10] of char;      char varid[11];
```

The types ''double'', ''single'', ''quadruple'', and ''unsigned'', are accepted and translated into corresponding C data types. For example:

```
Pascal                           C
------                           -
d: double;                       double d;
s: single;                       float s;
q: quadruple;                    double q;
```

```
u: unsigned;                          unsigned u;
```

Initializer values are allowed on variable declarations. The Pascal syntax is:

```
Pascal                                C
------                                -
varid: integer := const-expr;         int varid = const-expr;
```

Attributes associated with type and variable declarations and procedure or function declarations are allowed. The attributes are simply discarded in the C translation. The accepted syntax is:

```
type_id = [attribute-list] type;
variable_id: [attribute-list] type;
[attribute-list] procedure name(args);
```

External Procedures

The ptoc translator recognizes the keywords "external", "extern", "fortran", and "forward" on procedure and function declarations. In the C translation, this will simply generate a function type definition. The syntax is:

```
procedure name (params); external;
```

Mechanism-specifiers which are used to describe parameter attributes are recognized. The mechanism-specifiers (%ref, %descr, %stdescr, %immed) are simply discarded in the C translation. The accepted syntax is:

```
procedure name (%REF param1: integer := %IMMED 0); external;
```

Case Statement

The ptoc translator accepts the keyword "otherwise" as the default selection in a Pascal case statement. This is translated to "default" in the C switch statement.

Octal And Hexadecimal Numbers

The ptoc translator accepts octal and hex numbers as values for constant declarations and as valid numbers in expressions. The accepted syntax for constant declarations is:

```
CONST
    hexone = %x 'DEC';                (* hex const *)
    octone = %O '777';                (* octal const *)
```

The accepted syntax in expressions is:

```
i := %X'DEC';
if (i > %o'777')
then i := i * hexone + %x 'abc';
```

Operators

The ptoc translator accepts the operator "rem", and translates it the same as the "mod" operator. This produces the "%" operator in C.

Restrictions

Syntax

All Pascal source files and header files must not contain any Pascal syntax errors. The `ptoc` translator may dump core at runtime on some types of syntax errors in input files.

Arrays

Lower bounds of Pascal arrays are ignored. Only the upper bound is considered for the C translation. The C translation will declare the array with enough space to index from 0 through the upper bound of the Pascal array. A negative Pascal array bound will need special attention from the user, since C does not allow negative array bounds.

The `ptoc` translator does not handle array bounds which are enumerated "in place". Constructs like this will produce a syntax error:

```
A: array[(RED,WHITE,BLUE)] of integer;
```

The `ptoc` translator does not translate array declarations where the base type of the array is in turn a complex type. The base type of the array is set to "integer" and a warning is printed. The following Pascal declaration is an example:

```
A: array[1..10] of array[1..20] of char;
```

Pointer Declarations

Pascal allows type declarations of pointers to objects that are not defined yet. The `ptoc` translator translates these to C and generates a warning message. It is illegal in C for a pointer type to reference an object that is not defined yet, so this will require the user to modify the C source translation.

Empty Records

Pascal allows empty records. These translate to empty structures in C. Empty structures will produce syntax errors in C. The user must edit these in the Pascal file or in the C translation.

Subrange Declarations

Pascal allows types and variables of subrange types, which C does not. The `ptoc` translator translates these into types or variables of the base type of the Pascal subrange. For example:

```
Pascal                          C
------                          -
sr1: -10..10;                   int sr1;
sr2: 'a'..'z';                  char sr2;
```

Write Statements

The `ptoc` translator does not handle all possible forms of Pascal write statements with complex variables. If `ptoc` complains about the syntax of a write statement, comment it out and translate it to C by hand.

Sets

A Pascal set declaration becomes a plain variable in C, having the same type as the base type of the Pascal set. For example:

```
Pascal                  C
------                  -
v: set of char;         char v;
```

Pascal statements using certain set constructs will translate into C code that will not compile. You will have to comment out the Pascal code and translate by hand, or edit the C code to correct it. For example:

```
Pascal                  C
------                  -
v := ['a','b','c'];     v = ['a']['b']['c'];
```

The `ptoc` translator interprets the set construct as an array index and generates array index code.

Nested Procedures

Pascal nested procedures are linearized. The corresponding C functions are all at the same lexical level. Variables that are defined at an outer procedure scope level and are referenced by a procedure at a nested scope level, will be undefined in the C translation of the nested procedure. Such variables must be declared global to all procedures in the Pascal source file or they must be passed as arguments to the nested procedure.

Files

`/usr/bin/ptoc`

Name

ratfor – rational FORTRAN dialect

Syntax

ratfor [*option* ...] [*filename* ...]

Description

The `ratfor` command converts a rational dialect of FORTRAN into ordinary irrational FORTRAN. `ratfor` provides control flow constructs essentially identical to those in C:

statement grouping:
    { statement; statement; statement }

decision-making:
```
    if (condition) statement [ else statement ]
    switch (integer value) {
        case integer:    statement
        ...
        [ default: ]     statement
    }
```

loops:
```
    while (condition) statement
    for (expression; condition; expression) statement
    do limits statement
    repeat statement [ until (condition) ]
    break
    next
```

and some syntactic sugar to make programs easier to read and write:

free form input:
    multiple statements/line; automatic continuation

comments:
    # this is a comment

translation of relationals:
    >, >=, etc., become .GT., .GE., etc.

return (expression)
    returns expression to caller from function

define: define name replacement

include: include filename

`ratfor` is best used with `f77`(1).

See Also

f77(1)
B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, 1976.

Name

struct – structure FORTRAN programs

Syntax

struct [*options*] ... *file*

Description

The `struct` command translates the FORTRAN program specified by *file* (standard input default) into a `ratfor` program. Wherever possible, `ratfor` control constructs replace the original FORTRAN. Statement numbers appear only where still necessary. Cosmetic changes are made, including changing Hollerith strings into quoted strings and relational operators into symbols (for example, ".GT." into ">"). The output is appropriately indented.

Options

–s Input is accepted in standard format, that is comments are specified by a c, C, or * in column 1, and continuation lines are specified by a nonzero, nonblank character in column 6. Normally input is in the form accepted by `f77`(1).

–i Do not turn computed goto statements into switches. (`ratfor` does not turn switches back into computed goto statements.)

–a Turn sequences of else ifs into a non-`ratfor` switch of the form

```
switch
        {   case pred1: code
            case pred2: code
            case pred3: code
            default: code
        }
```

The case predicates are tested in order; the code appropriate to only one case is executed. This generalized form of switch statement does not occur in `ratfor`.

–b Generate goto's instead of multilevel break statements.

–n Generate goto's instead of multilevel next statements.

–t*n* Make the nonzero integer *n* the lowest valued label in the output program (default 10).

–c*n* Increment successive labels in the output program by the nonzero integer *n* (default 1).

–e*n* If *n* is 0 (default), place code within a loop only if it can lead to an iteration of the loop. If *n* is nonzero, admit a small code segments to a loop if otherwise the loop would have exits to several places including the segment, and the segment can be reached only from the loop. 'Small' is close to, but not equal to, the number of statements in the code segment.

Values of n under 10 are suggested.

Restrictions

Struct knows FORTRAN 66 syntax, but not full FORTRAN 77.
If an input FORTRAN program contains identifiers which are reserved words in `ratfor`, the structured version of the program will not be a valid `ratfor` program.
The labels generated cannot go above 32767.
If you get a goto without a target, try **–e .**

Files

/tmp/struct*
/usr/lib/struct/*

See Also

f77(1)

Name

sysline – display system status on status line of a terminal

Syntax

sysline [**–bcdehDilmpqrsj**] [**–H** *remote*] [**+N**]

Description

The `sysline` command runs in the background and periodically displays system status information on the status line of the terminal. Not all terminals contain a status line. Those that do include the h19, concept 108, Ann Arbor Ambassador, vt100, Televideo 925/950 and Freedom 100. If no options are given, `sysline` displays the time of day, the current load average, the change in load average in the last 5 minutes, the number of users (followed by a 'u'), the number of runnable process (followed by a 'r')(VAX only), the number of suspended processes (followed by a 's')(VAX only), and the users who have logged on and off since the last status report. Finally, if new mail has arrived, a summary of it is printed. If there is unread mail in your mailbox, an asterisk will appear after the display of the number of users. The display is normally in reverse video (if your terminal supports this in the status line) and is right justified to reduce distraction. Every fifth display is done in normal video to give the screen a chance to rest.

If you have a file named `.who` in your home directory, then `sysline` prints the contents of that file first. One common use of this feature is to alias chdir, pushd, and popd to place the current directory stack in `~/.who` after it changes the new directory.

Options

–b Beep once every half hour and twice every hour, just like those obnoxious watches you keep hearing.

–c Clear the status line for 5 seconds before each redisplay.

–d Debug mode -- print status line data in human readable format.

–D Print out the current day/date before the time.

–e Print out only the information. Do not print out the control commands necessary to put the information on the bottom line. This option is useful for putting the output of `sysline` onto the mode line of an emacs window.

–H *remote* Print the load average on the remote host *remote* (VAX only). If the host is down, or is not sending out *rwhod* packets, then the down time is printed instead.

–h Print out the host machine's name after the time (VAX only).

–l Don't print the names of people who log in and out.

–m Don't check for mail.

–p Don't report the number of processes which are runnable and suspended.

–r	Don't display in reverse video.
+N	Update the status line every N seconds. The default is 60 seconds.
–q	Don't print out diagnostic messages if something goes wrong when starting up.
–i	Print out the process id of the `sysline` process onto standard output upon startup. With this information you can send the alarm signal to the `sysline` process to cause it to update immediately. The `sysline` command writes to the standard error, so you can redirect the standard output into a file to catch the process id.
–s	Print "short" form of line by left-justifying. No *iff* escapes are allowed in the status line. Some terminals (the Televideos and Freedom 100 for example) do not allow cursor movement (or other "intelligent" operations) in the status line. For these terminals, `sysline` normally uses blanks to cause right-justification. This flag will disable the adding of the blanks.
–j	Force `sysline` output to be left-justified, even on terminals capable of cursor movement on the status line.

If you have a file .syslinelock in your home directory, then `sysline` will not update its statistics and write on your screen; it will just go to sleep for a minute. This is useful if you want to momentarily disable `sysline`. Note that it may take a few seconds from the time the lock file is created until you are guaranteed that `sysline` will not write on the screen. In order to use `sysline` on a VT100 terminal, either of the following termcap entries should be used to specify terminal attributes:

vt100-s	Creates a top-of-screen status line.
vt100-s-bot	Creates a bottom of screen status line.

Restrictions

If you interrupt the display, then you may find your cursor missing or stuck on the status line. The best thing to do is reset your terminal. If there is too much output for one line, the excess is thrown away.

Files

/etc/utmp	Names of people who are logged in.
/dev/kmem	Process table (VAX only).
/usr/spool/rwho/whod.*	Who/uptime information for remote hosts (VAX only).
print	
HOME/.whoInformationonbottomline..TP{HOME}/.syslinelock	When the .syslinelock file exists, `sysline` will not print.

Name

tc – phototypesetter simulator

Syntax

tc [**–t**] [**–s***N*] [**–p***L*] [*file*]

Description

The `tc` command interprets its input (standard input default) as device codes for a Graphic Systems phototypesetter (cat). The standard output of `tc` is intended for a Tektronix 4015 (a 4014 terminal with ASCII and APL character sets). The sixteen typesetter sizes are mapped into the 4014's four sizes; the entire TROFF character set is drawn using the 4014's character generator, using overstruck combinations where necessary. Typical usage:

troff –t file | tc

At the end of each page *tc* waits for a newline (empty line) from the keyboard before continuing on to the next page. In this wait state, the command **e** will suppress the screen erase before the next page; **s**N will cause the next N pages to be skipped; and **!**line will send line to the shell.

Options

–t	Don't wait between pages; for directing output into a file.
–s*N*	Skip the first *N* pages.
–p*L*	Set page length to *L*. *L* may include the scale factors **p** (points), **i** (inches), **c** (centimeters), and **P** (picas); default is picas.
'–l w'	Multiply the default aspect ratio, 1.5, of a displayed page by *l*/*w*.

Restrictions

Font distinctions are lost.

`tc`'s character set is limited to ASCII in just one size.

See Also

troff(1), plot(1g)

tk(1)
Unsupported

Name

tk – paginator for the Tektronix 4014

Syntax

tk [**–t**] [*–N*] [**–p***L*] [*file*]

Description

The output of `tk` is intended for a Tektronix 4014 terminal. `tk` arranges for 66 lines to fit on the screen, divides the screen into *N* columns, and contributes an eight space page offset in the (default) single-column case. Tabs, spaces, and backspaces are collected and plotted when necessary. Teletype Model 37 half- and reverse-line sequences are interpreted and plotted. At the end of each page `tk` waits for a newline (empty line) from the keyboard before continuing on to the next page. In this wait state, the command *!command* will send the *command* to the shell.

Options

–t	Don't wait between pages; for directing output into a file.
–N	Divide the screen into *N* columns and wait after the last column.
–p*L*	Set page length to *L* lines.

See Also

pr(1)

Name

tp – manipulate tape archive

Syntax

tp [*key*] [*name...*]

Description

The `tp` command saves and restores files to and from an archive on DECtape or magnetic tape. The default archive is `tapx`, but any file or device may be requested through the use of options. Its actions are controlled by the *key* argument. The key is a string of characters containing at most one function letter and possibly one or more function modifiers. Other arguments to the command are file or directory names specifying which files are to be dumped, restored, or listed. In all cases appearance of a directory name refers to the files and (recursively) subdirectories of that directory.

NOTE

The DECtape device is not supported in ULTRIX.

The function portion of the key is specified by one of the following letters:

d Deletes the named files from the tape. At least one *name* argument must be given. This function is not permitted on magnetic tapes.

r The named files are written on the tape. If files with the same names already exist, they are replaced. "Same" is determined by string comparison, so `./abc` is not the same as `/usr/dmr/abc` even if `/usr/dmr` is the current directory. If no *name* argument is given, '.' is the default.

t Lists the names of the specified files. If no *name* argument is given, the entire contents of the tape are listed.

u Updates the tape. **u** is like **r,** but a file is replaced only if its modification date is later than the date stored on the tape. **u** is the default function key if none is given.

x Extracts the named files from the tape to the file system. The owner and mode are restored. If no *name* argument is given, the entire contents of the tape are extracted.

You can use one or more of the following options in addition to the letter that selects the function desired.

c A fresh dump is to be created. The tape directory is cleared before beginning. This option is usable only with the **r** and **u** function keys. This option is assumed with magnetic tape since it is impossible to selectively overwrite magnetic tape.

f Use the next argument as the name of the archive. Any special file can be used as the next argument. When used with the **r** function key, the **c** option is implied. That is, the directory is cleared before beginning.

This option cannot be used with the **d** or **u** function keys.

i Errors reading and writing the tape are noted, but no action is taken. Normally, errors cause a return to the command level.

m Specifies magnetic tape as opposed to DECtape. The default tape is `rmt0h`, but other devices may be requested with the **f** option.

When used with the **r** function key, the **c** option is implied. That is, the directory is cleared before beginning.

This option cannot be used with the **d** or **u** function keys.

v Normally, `tp` does its work silently. This option causes it to display the name of each file it treats, preceded by the function letter. With the **t** function, **v** gives more information about the entries than just the name.

w Causes *tp* to pause before transferring each file, type the indicative letter and the file name (as with **v**) and await the user's response.

Response **y** means 'yes', so the file is transferred. Null response means 'no', and the file is not transferred

Response **x** means "exit"; the *tp* command terminates immediately. With the **x** function key, files previously asked about have been extracted already. With the **r**, **u**, and **d** function keys, no change has been made to the tape.

Restrictions

A single file with several links to it is treated like several files.

Binary-coded control information makes magnetic tapes written by `tp` difficult to carry to other machines. The `tar`(1) command avoids this problem.

Diagnostics

There are several. The least obvious one is 'Phase error', which means the file changed after it was selected for dumping but before it was dumped.

Files

`/dev/tap?` DECtape archive

`/dev/rmt?h` or `/dev/nmt?h`
Magnetic tape archive

See Also

ar(1), tar(1)

Name

trman – translate version 6 manual macros to version 7 macros

Syntax

trman [*file*]

Description

The `trman` command reads the input *file*, which should be nroff/troff input and attempts to translate the version 6 manual sections therein to version 7 format. It is largely successful, but seems to have trouble with indented paragraphs and complicated font control. You should expect to have to fix up long sections by hand somewhat.

See Also

man(7)

troff(1)
Unsupported

Name

troff – text formatting and typesetting

Syntax

troff [*option*] ... [*file*] ...

Description

The `troff` program formats text in the named *files* for printing on a Graphic Systems C/A/T phototypesetter; `nroff` is used for for typewriter-like devices. Their capabilities are described in the *Nroff/Troff User's Manual.*

If no *file* argument is present, the standard input is read. An argument consisting of a single minus (–) is taken to be a file name corresponding to the standard input.

Options

The options, which may appear in any order so long as they appear before the files, are:

–o*list* Print only pages whose page numbers appear in the comma-separated *list* of numbers and ranges. A range *N–M* means pages *N* through *M*; an initial *–N* means from the beginning to page *N*; and a final *N–* means from *N* to the end.

–n*N* Number first generated page *N*.

–s*N* Stop every *N* pages. *Nroff* will halt prior to every *N* pages (default *N*=1) to allow paper loading or changing, and will resume upon receipt of a newline. `troff` will stop the phototypesetter every *N* pages, produce a trailer to allow changing cassettes, and resume when the typesetter's start button is pressed.

–m*name* Prepend the macro file **/usr/lib/tmac/tmac.***name* to the input *files*.

–r*aN* Set register *a* (one-character) to *N*.

–i Read standard input after the input files are exhausted.

–q Invoke the simultaneous input-output mode of the **rd** request.

troff Only

–t Direct output to the standard output instead of the phototypesetter.

–f Refrain from feeding out paper and stopping phototypesetter at the end of the run.

–w Wait until phototypesetter is available, if currently busy.

–b Report whether the phototypesetter is busy or available. No text processing is done.

–a Send a printable ASCII approximation of the results to the standard output.

–pN Print all characters in point size *N* while retaining all prescribed spacings and motions, to reduce phototypesetter elapsed time.

–F*fontdir*
The directory *fontdir contains the font width tables* /usr/lib/font. This option can be used to produce output for devices besides the phototypesetter.

If the file */usr/adm/tracct* is writable, `troff` keeps phototypesetter accounting records there. The integrity of that file may be secured by making `troff` a 'set user-id' program.

Files

/tmp/ta*	temporary file
/usr/lib/tmac/tmac.*	standard macro files
/usr/lib/term/*	terminal driving tables for `nroff`
/usr/lib/font/*	font width tables for `troff`
/dev/cat	phototypesetter
/usr/adm/tracct	accounting statistics for /dev/cat

See Also

eqn(1), tbl(1), ms(7), me(7), man(7), col(1)
J. F. Ossanna, *Nroff/Troff user's manual*
B. W. Kernighan, *A TROFF Tutorial*

unifdef(1)
Unsupported

Name

unifdef – removes ifdef'ed lines

Syntax

unifdef [**–c** **–D***sym* **–id***sym* **–iu***sym* **–l** **–t** **–U***sym*] ... [*file*]

Description

The `unifdef` command is useful for removing ifdef'ed lines from a file while otherwise leaving the file alone. The `unifdef` command is like a stripped-down C preprocessor: it is smart enough to deal with the nested ifdefs, comments, single and double quotes of C syntax so that it can do its job, but it doesn't do any including or interpretation of macros. Neither does it strip out comments, though it recognizes and ignores them. Ifdefs involving symbols you don't specify are untouched and copied out along with their associated ifdef, else, and endif lines. The ifdef, ifndef, else, and endif lines associated with the symbol, *sym*, will also be removed. If an ifdef X occurs nested inside another ifdef X, then the inside ifdef is treated as if it were an unrecognized symbol. If the same symbol appears in more than one argument, only the first occurrence is significant.

If you use ifdefs to delimit non-C lines, such as comments or code which is under construction, then you must tell *unifdef* which symbols are used for that purpose so that it won't try to parse for quotes and comments in those ifdef'ed lines.

The `unifdef` command copies its output to stdout and will take its input from stdin if no *file* argument is given.

Options

–c Causes the operation of `unifdef` to be complemented. The lines that would have been removed or blanked are retained and vice versa.

–id*sym* Specifies that you want the lines inside certain ifdefs to be ignored, but copied out.

–iu*sym* Specifies that you want the lines inside certain ifdefs to be ignored, but copied out.

–l Causes `unifdef` to replace removed lines with blank lines instead of deleting them.

–t Makes `unifdef` refrain from attempting to recognize comments and single and double quotes. This option is for use on plain text (not C code).

–D*sym* Specifies which symbols you want defined and the lines inside those ifdefs will be copied to the output.

–U*sym* Specifies which symbols you want undefined and the lines inside those ifdefs will be removed.

Restrictions

Does not know how to deal with *cpp* constructs such as

```
#if     defined(X) || defined(Y)
```

Diagnostics

Premature EOF, inappropriate else or endif.

Exit status is 0 if output is exact copy of input, 1 if not, 2 if trouble.

See Also

diff(1)

Name

units – conversion program

Syntax

units

Description

The `units` command converts quantities expressed in various standard scales to their equivalents in other scales. It works interactively in this fashion:

You have: inch
You want: cm
* *2.54000e+00*
/ *3.93701e–01*

A quantity is specified as a multiplicative combination of units optionally preceded by a numeric multiplier. Powers are indicated by suffixed positive integers, division by the usual sign:

You have: 15 pounds force/in2
You want: atm
* *1.02069e+00*
/ *9.79730e–01*

The `units` command only does multiplicative scale changes. Thus it can convert Kelvin to Rankine, but not Centigrade to Fahrenheit. Most familiar units, abbreviations, and metric prefixes are recognized, together with a generous leavening of exotica and a few constants of nature including:

pi	ratio of circumference to diameter
c	speed of light
e	charge on an electron
g	acceleration of gravity
force	same as g
mole	Avogadro's number
water	pressure head per unit height of water
au	astronomical unit

'Pound' is a unit of mass. Compound names are run together, for example, 'lightyear'. British units that differ from their US counterparts are prefixed thus: 'brgallon'. Currency is denoted 'belgiumfranc', 'britainpound', ...

If `units` is used with no arguments, it will use `/usr/lib/units` to define the units and the conversions that it knows about. When *file* is supplied, it will be used to define the units and conversions that `units` will recognize.

Files

`/usr/lib/units`

Name

vfontinfo – inspect and print out information about UNIX fonts

Syntax

vfontinfo [**–v**] *fontname* [*characters*]

Description

The `vfontinfo` command allows you to examine a font in the UNIX format. It prints out all the information in the font header and information about every non-null (width > 0) glyph. This can be used to make sure the font is consistent with the format.

The *fontname* argument is the name of the font you wish to inspect. It writes to standard output. If it can't find the file in your working directory, it looks in `/usr/lib/vfont` (the place most of the fonts are kept).

The *characters*, if given, specify certain characters to show. If omitted, the entire font is shown.

Options

–v Displays the bits of the glyph itself as an array of X's and spaces, in addition to the header information.

See Also

vpr(1), vfont(5)
The Berkeley Font Catalog

Name

vgrind – grind nice listings of programs

Syntax

vgrind [**–f**] [**–**] [**–t**] [**–n**] [**–x**] [**–W**] [**–sn**] [**–h** *header*] [**–d** file] [**–l***language*] *name* ...

Description

The `vgrind` command formats the program sources which are arguments in a nice style using `troff`(1). Comments are placed in italics, keywords in bold face, and the name of the current function is listed down the margin of each page as it is encountered.

`vgrind` runs in two basic modes, filter mode or regular mode. In filter mode `vgrind` acts as a filter in a manner similar to `tbl1` The standard input is passed directly to the standard output except for lines bracketed by the `troff`-like macros:

.vS - starts processing

.vE - ends processing

These lines are formatted as described above. The output from this filter can be passed to `troff` for output. There need be no particular ordering with `eqn`(1) or `tbl`(1).

In regular mode, `vgrind` accepts input files, processes them, and passes them to `troff`(1) for output.

In both modes `vgrind` passes any lines beginning with a decimal point without conversion.

Options

–f Forces filter mode

– Forces input to be taken from standard input (default if **–f** is specified)

–t Similar to the same option in `troff` causing formatted text to go to the standard output

–n Forces no keyword bolding

–x Outputs the index file in a ''pretty'' format. The index file itself is produced whenever `vgrind` is run with a file called `index` in the current directory. The index of function definitions can then be run off by giving `vgrind` the **–x** option and the file `index` as argument.

–W Forces output to the (wide) Versatec printer rather than the (narrow) Varian

–s Specifies a point size to use on output (exactly the same as the argument of a .ps)

–h *header*
Specifies a particular *header* to put on every output page (default is the file name)

–d Specifies an alternate language definitions *file* (default is /usr/lib/vgrindefs)

–l *language*
Specifies the *language* to use. Currently known are PASCAL (**–lp**), MODEL (**–lm**),**C** (**–lc** or the default), CSH (**–lcsh**), SHELL (**–lsh**), RATFOR (**–lr**), and ICON (**–ll**).

Restrictions

Vfontedpr assumes that a certain programming style is followed:

For **C** – function names can be preceded on a line only by spaces, tabs, or an asterisk. The parenthesized arguments must also be on the same line.

For **PASCAL** – function names need to appear on the same line as the keywords *function* or *procedure*.

For **MODEL** – function names need to appear on the same line as the keywords *is beginproc*.

If these conventions are not followed, the indexing and marginal function name comment mechanisms will fail.

More generally, arbitrary formatting styles for programs mostly look bad. The use of spaces to align source code fails miserably; if you plan to `vgrind` your program you should use tabs. This is somewhat inevitable since the font used by `vgrind` is variable width.

Files

index	file where source for index is created
/usr/lib/tmac/tmac.vgrind	macro package
/usr/lib/vfontedpr	preprocessor
/usr/lib/vgrindefs	language descriptions

See Also

vlp(1), vtroff(1), vgrindefs(5)

Name

vlp – Format Lisp programs to be printed with nroff, vtroff, or troff

Syntax

vlp [**–p** *pointsize*] [**–d**] [**–f**] [**–l**] [**–v**] [**–T** *title1*] *file1* [**–T** *title2*] *file2* ...

Description

The `vlp` command formats the named *files* so that they can be run through `nroff`, `vtroff`, or `troff` to produce listings that line-up and are attractive. The first non-blank character of each line is lined-up vertically, as in the source file. Comments (text beginning with a semicolon) are printed in italics. Each function's name is printed in bold face next to the function. This format makes Lisp code look attractive when it is printed with a variable width font.

Normally, `vlp` works as a filter and sends its output to the standard output. However, the **–v** option pipes the output directly to `vtroff`. If no *files* are specified, then `vlp` reads from the standard input.

Options

–p Changes the size of the text from its default value of 8 points to one of 6, 8, 10, or 12 points. Once set, the point size is used for all subsequent files. This point size does not apply to embedded text (see the **–f** option).

–d Puts `vlp` into debugging mode.

–f Sets the filtered mode in which all lines are passed unmodified, except those lines between the directives **.Ls** and **.Le**. This mode can be used to format Lisp code that is embedded in a document. The directive **.Ls** takes an optional argument that gives the point size for the embedded code. If not size is specified, the size of the surrounding text is used.

–l Prevents `vlp` from placing labels next to functions. This switch is useful for embedded Lisp code, where the labels would be distracting.

–v Causes `vlp` to send its output to `vtroff` rather than the standard output.

–T Print a title on each page. The **–T** option applies only to the next *file* given. Titles are not printed for embedded text (see **–f**, above). This option may not be used if `vlp` is reading from the standard input.

Restrictions

`vlp` transforms \ into \\ so that it will be printed out. Hence, `troff` commands cannot be embedded in Lisp code.

Files

/usr/lib/vlpmacs	troff/nroff macros

See Also

vgrind(1), lisp(1)

VAX **vpr(1)**
Unsupported

Name

vpr, vprm, vpq, vprint – raster printer/plotter spooler

Syntax

vpr [**–W**] [**–l**] [**–v**] [**–t** [**–1234** font]] [**–w**] [**–w***width*] [**–m**] [*file* ...]
vprm [id ...] [*file* ...] [*owner* ...]
vpq
vprint [**–W**] *file* ...

Description

The `vpr` spooler causes the named *files* to be queued for printing or typeset simulation on one of the available raster printer/plotters. If no *files* are named, the standard input is read. By default the input is assumed to be line printer-like text. For very wide plotters, the input is run through the filter `/usr/lib/sidebyside` giving it an argument of **–w106** which arranges it four pages adjacent with 90 column lines (the rest is for the left margin). Since there are 8 lines per inch in the default printer font, `vpr` thus produces 86 lines per page (the top and bottom lines are left blank).

Options

–l Print the input in a more literal manner. Page breaks are not inserted, and most control characters (except format effectors: \n, \f, etc.) are printed (many control characters print special graphics not in the ASCII character set.) Tab and underline processing is still done. If this option is not given, control characters which are not format effectors are ignored, and page breaks are inserted after an appropriate number of lines have been printed on a page.

–W Queues files for printing on a wide output device, if available. Normally, files are queued for printing on a narrow output device.

–1234 Specifies a font to be mounted on font position *i*. The daemon will construct a `.railmag` file referencing `/usr/lib/vfont/name.size`.

–m Report by *mail*(1) when printing is complete.

–w (Applicable only to wide output devices.) Do not run the input through sidebyside. Such processing has been done already, or full (440 character) printer width is desired.

–w*width* Use width *width* rather than 90 for *sidebyside*.

–v Use the filter `/usr/lib/vrast` to convert the vectors to raster. The named files must be a parameter and vector file (in that order) created by *plot*(3x) routines.

–t Use the filter `/usr/lib/vcat` to typeset the input on the printer/plotter. The input must have been generated by `troff`(1) run with the **–t** option. This is not normally run directly to wide output devices, since it is wasteful to run only one page across.

The program `vtroff(1)` is normally used and arranges, using `vsort` for printing to occur four pages across, conserving paper.

`vprm` removes entries from the raster device queues. The id, filename or owner should be that reported by `vpq`. All appropriate files will be removed. Both queues are always searched. The id of each file removed from the queue will be printed.

`vpq` prints the queues. Each entry in the queue is printed showing the owner of the queue entry, an identification number, the size of the entry in characters, and the file which is to be printed. The *id* is useful for removing a specific entry from the printer queue using `vprm`.

`vprint` is a shell script which pr's a copy of each named file on one of the electrostatic printer/plotters. The files are normally printed on a narrow device; **-W** option causes them to be printed on a wide device.

Restrictions

The 1's (one's) and l's (lower-case el's) in a Benson-Varian's standard character set look very similar; caution is advised.

Files

/usr/spool/v?d/*	device spool areas
/usr/lib/v?d	daemons
/usr/lib/vpd	Versatec daemon
/usr/lib/vpf	filter for printer simulation
/usr/lib/*vcat	filter for typeset simulation
/usr/lib/vrast	filter for plot
/usr/lib/sidebyside	filter for wide output

See Also

troff(1), vfont(5), vp(4), pti(1), vtroff(1), plot(3x)

VAX **vtroff(1)**
Unsupported

Name

vtroff – troff to a raster plotter

Syntax

vtroff [**–w**] [**–F** majorfont] [**–123** minorfont] [**–l***length*] [**–x**] "*troff arguments*"

Description

The `vtroff` command runs `troff`(1) sending its output through various programs to produce typeset output on a raster plotter such as a Benson-Varian or or a Versatec. The **–W** option specifies that a wide output device be used; the default is to use a narrow device. The **–l** (lower case l) option causes the output to be split onto successive pages every *length* inches rather than the default 11''.

The default font is a Hershey font. If some other font is desired you can give a **–F** argument and then the font name. This will place normal, italic and bold versions of the font on positions 1, 2, and 3. To place a font only on a single position, you can give an argument of the form *–n* and the minor font name. A **.r** will be added to the minor font name if needed. Thus

% **vtroff –ms** *paper*

will set a paper in the Hershey font, while

% **vtroff –F nonie –ms** *paper*

will set the paper in the (sans serif) nonie font. The **–x** option asks for exact simulation of photo-typesetter output (that is, using the width tables for the C.A.T. photo-typesetter).

Restrictions

Since some macro packages work correctly only if the fonts named R, I, B, and S are mounted, and since the Versatec fonts have different widths for individual characters than the fonts found on the typesetter, the following dodge was necessary: If you don't use the ''.fp'' `troff` directive then you get the widths of the standard typesetter fonts suitable for shipping the output of troff over the network to the computer center A machine for phototypesetting. If, however, you remount the R, I, B and S fonts, then you get the width tables for the Versatec.

Files

/usr/lib/tmac/tmac.vcat	default font mounts and bug fixes
/usr/lib/fontinfo/*	fixes for other fonts
/usr/lib/vfont	directory containing fonts

See Also

troff(1), vfont(5), vpr(1)

Name

vwidth – make troff width table for a font

Syntax

vwidth *fontfile pointsize* > ftxx.c
cc -c ftxx.c mv ftxx.o /usr/lib/font/ftxx

Description

The `vwidth` command translates from the width information stored in the vfont style format to the format expected by `troff`. `troff` wants an object file in a.out(5) format. (This fact does not seem to be documented anywhere.) `troff` should look directly in the font file but it doesn't.

`vwidth` should be used after editing a font with `fed`(1). It is not necessary to use `vwidth` unless you have made a change that would affect the width tables. Such changes include numerically editing the width field, adding a new character, and moving or copying a character to a new position. It is *not* always necessary to use `vwidth` if the physical width of the glyph (for example, the number of columns in the bit matrix) has changed, but if it has changed much the logical width should probably be changed and `vwidth` run.

`vwidth` produces a C program on its standard output. This program should be run through the C compiler and the object (that is, the .o file) saved. The resulting file should be placed in `/usr/lib/font` in the file ft*xx* where is a one or two letter code that is the logical (internal to `troff`) font name. This name can be found by looking in the file /usr/lib/fontinfo/*fname** where *fname* is the external name of the font.

Restrictions

Produces the C file using obsolete syntax about which the portable C compiler complains.

See Also

fed(1), vfont(5), troff(1), vtroff(1)

whois(1)
Unsupported

Name

whois – DARPA Internet user name directory service

Syntax

whois [**–h** *servername*] *name*

Description

The `whois` command allows you to look up people, hosts, and organizations in the database kept by the Network Information Center (NIC) at SRI International. The *name* can be the last name of a registered user, or the name of a registered Internet host, or other things recognized by the whois server. Not all users, and only a few Internet hosts, are registered with the NIC. If the *name* argument contains white space or other special characters, you must surround it with double quote (") marks to prevent its interpretation by the shell.

The WHOIS server at the NIC will provide more information on doing complex searches if you do

```
whois "?"
```

(note the quotes around the question mark), or even more information if you do

```
whois help
```

Options

–h *servername* Specifies the whois server on a host other than the default (NIC.DDN.MIL).

Examples

```
whois cerf
whois dec.com
whois -h nic.ddn.mil knuth
whois smith
```

Note that the last example, at least, will match many records (and may take fairly long to complete). The whois server at the NIC will automatically provide information on how to single out one record. Remember to put quotes around strings that contain '!' characters.

See Also

RFC 812: Nicname/Whois

Name

dial – establish an out-going terminal line connection

Syntax

#include <dial.h>

int dial(*call*)
CALL **call*;

void undial(*fd*)
int *fd*;

Description

The `dial` routine returns a file-descriptor for a terminal line open for read/write. The argument to `dial` is a CALL structure which is defined in the <dial.h> header file.

When finished with the terminal line, the calling program must invoke `undial` to release the semaphore that has been set during the allocation of the terminal device.

The CALL typedef in the <dial.h> header file is:

```
typedef struct {
       struct termio *attr;/* pointer to termio attribute struct */
       int    baud;  /* transmission data rate */
       int    speed; /* 212A modem: low=300, high=1200 */
       char   *line; /* device name for out-going line */
       char   *telno;       /* pointer to tel-no digits string */
       int    modem; /* specify modem control for direct lines */
} CALL;
```

The CALL element `speed` is intended only for use with an outgoing dialed call.

The CALL element `baud` is for the desired transmission baud rate. For example, one might set baud to 110 and speed to 300 (or 1200).

If the desired terminal line is a direct line, a string pointer to its device-name should be placed in the `line` element in the CALL structure. Legal values for such terminal device names are kept in the L-devices file. In this case, the value of the `baud` element need not be specified as it is determined from the L-devices file.

The `telno` element is for a pointer to a character string representing the telephone number to be dialed. The termination symbol is supplied by the `dial` function, and should not be included in the `telno` string passed to `dial` in the CALL structure.

The CALL element `modem` is used to specify modem control for direct lines. This element should be non-zero if modem control is required.

The CALL element `attr` is a pointer to a `termio` structure, as defined in the termio.h header file. A NULL value for this pointer element may be passed to the `dial` function, but if such a structure is included, the elements specified in it are set for the outgoing terminal line before the connection is established. This is often important for certain attributes such as parity and baud-rate.

Restrictions

Including the **<dial.h>** header file automatically includes the **<termio.h> header file.**

An `alarm`(3) system call for 3600 seconds is made (and caught) within the `dial` module for the purpose of "touching" the *LCK..* file and constitutes the device allocation semaphore for the terminal device. Otherwise, `uucp`(1c) may simply delete the *LCK..* entry on its 90-minute clean-up rounds. The alarm may go off while the user program is in a `read`(2) or `write`(2) system call, causing an apparent error return. If the user program expects to be around for an hour or more, error returns from *read*s should be checked for **(errno==EINTR)**, and the *read* possibly reissued.

Diagnostics

On failure, a negative value indicating the reason for the failure is returned. Mnemonics for these negative indices as listed here are defined in the <dial.h> header file.

```
INTRPT    -1    /* interrupt occured */
D_HUNG    -2    /* dialer hung (no return from write) */
NO_ANS    -3    /* no answer within 10 seconds */
ILL_BD    -4    /* illegal baud-rate */
A_PROB    -5    /* acu problem (open() failure) */
L_PROB    -6    /* line problem (open() failure) */
NO_Ldv    -7    /* can't open LDEVS file */
DV_NT_A   -8    /* requested device not available */
DV_NT_E   -12   /* requested speed does not match */
DV_NT_K   -9    /* requested device not known */
NO_BD_A   -10   /* no device available at requested baud */
NO_BD_K   -11   /* no device known at requested baud */
```

Files

`/usr/lib/uucp/L-devices`
`/usr/spool/uucp/LCK..`*tty-device*

See Also

uucp(1c), alarm(3), read(2), write(2), termio(7)

Name

intro – introduction to FORTRAN library functions

Description

This section describes those functions that are in the FORTRAN run time library. The functions listed here provide an interface from `f77` programs to the system in the same manner as the C library does for C programs. They are automatically loaded as needed by the FORTRAN compiler `f77`(1).

Most of these functions are in libU77.a. Some are in libF77.a or libI77.a. A few intrinsic functions are described for the sake of completeness.

For efficiency, the SCCS ID strings are not normally included in the `a.out` file. To include them, simply declare the following in any `f77` module:

```
external f77lid
```

VAX

abort(3f)
Unsupported

Name

abort – terminate abruptly with memory image

Syntax

subroutine abort (string)
character*(*) string

Description

The `abort` subroutine cleans up the I/O buffers and then aborts producing a *core* file in the current directory. If *string* is given, it is written to logical unit 0 preceded by "abort:".

Files

/usr/lib/libF77.a

See Also

abort(3)

Name

access – determine accessibility of a file

Syntax

integer function access (name, mode)
character*(*) name, mode

Description

The `access` subroutine checks the given file, *name*, for accessibility with respect to the caller according to *mode*. The *mode* argument may include in any order and in any combination one or more of:

r test for read permission
w test for write permission
x test for execute permission
(blank) test for existence

An error code is returned if either argument is illegal, or if the file can not be accessed in all of the specified modes. 0 is returned if the specified access would be successful.

Restrictions

Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.

Files

/usr/lib/libU77.a

See Also

access(2), perror(3f)

VAX **alarm (3f)**
Unsupported

Name

alarm – execute a subroutine after a specified time

Syntax

integer function alarm (time, proc)
integer time
external proc

Description

This routine arranges for subroutine *proc* to be called after *time* seconds. If *time* is "0", the alarm is turned off and no routine will be called. The returned value will be the time remaining on the last alarm.

Restrictions

Both `alarm` and `sleep` interact. If `sleep` is called after `alarm`, the `alarm` process will never be called. SIGALRM will occur at the lesser of the remaining `alarm` time or the `sleep` time.

Files

/usr/lib/libU77.a

See Also

alarm(3), sleep(3f), signal(3f)

Name

bessel functions – of two kinds for integer orders

Syntax

function besj0 (x)

function besj1 (x)

function besjn (n, x)

function besy0 (x)

function besy1 (x)

function besyn (n, x)

double precision function dbesj0 (x)
double precision x

double precision function dbesj1 (x)
double precision x

double precision function dbesjn (n, x)
double precision x

double precision function dbesy0 (x)
double precision x

double precision function dbesy1 (x)
double precision x

double precision function dbesyn (n, x)
double precision x

Description

These functions calculate Bessel functions of the first and second kinds for real arguments and integer orders.

Diagnostics

Negative arguments cause *besy0, besy1,* and *besyn* to return a huge negative value. The system error code will be set to EDOM (33).

Files

/usr/lib/libF77.a

See Also

bessel(3m), perror(3f)

Name

bit, and, or, xor, not, rshift, lshift – bitwise functions

Syntax

(intrinsic) function and (word1, word2)

(intrinsic) function or (word1, word2)

(intrinsic) function xor (word1, word2)

(intrinsic) function not (word)

(intrinsic) function rshift (word, nbits)

(intrinsic) function lshift (word, nbits)

Description

These bitwise functions are built into the compiler and return the data type of their argument(s). It is recommended that their arguments be *integer* values. Inappropriate manipulation of *real* objects may cause unexpected results.

The bitwise combinatorial functions return the bitwise "and" (`and`), "or" (`or`), or "exclusive or" (`xor`) of two operands. The `not` returns the bitwise complement of its operand.

The `lshift`, or `rshift` with a negative *nbits*, is a logical left shift with no end around carry. The `rshift`, or `lshift` with a negative *nbits*, is an arithmetic right shift with sign extension. No test is made for a reasonable value of *nbits*.

Files

These functions are generated in-line by the f77 compiler.

VAX

chdir(3f)
Unsupported

Name

chdir – change default directory

Syntax

integer function chdir (dirname)
character*(*) dirname

Description

The default directory for creating and locating files will be changed to *dirname*.

Restrictions

Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.

Use of this function may cause `inquire` by unit to fail.

Return Value

Zero is returned if successful; an error code otherwise.

Files

/usr/lib/libU77.a

See Also

cd(1), chdir(2), perror(3f)

Name

chmod – change mode of a file

Syntax

integer function chmod (name, mode)
character*(*) name, mode

Description

This function changes the filesystem *mode* of file *name*. The *mode* can be any specification recognized by `chmod`(1). The *name* must be a single pathname.

The normal returned value is 0. Any other value will be a system error number.

Restrictions

Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.

Files

/usr/lib/libU77.a
/bin/chmod exec'ed to change the mode.

See Also

chmod(1)

VAX **etime(3f)**
Unsupported

Name

etime, dtime – return elapsed execution time

Syntax

function etime (tarray)
real tarray(2)

function dtime (tarray)
real tarray(2)

Description

These two routines return elapsed runtime in seconds for the calling process. The `dtime` routine returns the elapsed time since the last call to `dtime`, or the start of execution on the first call.

The argument array returns user time in the first element and system time in the second element. The function value is the sum of user and system time.

The resolution of all timing is 1/HZ sec. where HZ is currently 60.

Files

/usr/lib/libU77.a

See Also

time(3f)

Name

exit – terminate process with status

Syntax

subroutine exit (status)
integer status

Description

The `exit` function flushes and closes all the process's files, and notifies the parent process if it is executing a `wait`. The low-order 8 bits of *status* are available to the parent process. Therefore, *status* should be in the range 0 – 255.

This call will never return.

The C function `exit` may cause cleanup actions before the final 'sys exit'.

Files

/usr/lib/libF77.a

See Also

exit(2), fork(2), wait(2), fork(3f), wait(3f)

VAX **fdate(3f)**
Unsupported

Name

fdate – return date and time in an ASCII string

Syntax

subroutine fdate (string)
character*(*) string

character*(*) function fdate()

Description

The `fdate` function returns the current date and time as a 24-character string in the format described under `ctime`(3). Neither 'newline' nor NULL will be included.

The `fdate` function can be called either as a function or as a subroutine. If called as a function, the calling routine must define its type and length. For example:

```
character*24    fdate
external        fdate

write(*,*) fdate()
```

Files

/usr/lib/libU77.a

See Also

ctime(3), idate(3f), time(3f)

Name

flmin, flmax, ffrac, dflmin, dflmax, dffrac, inmax – return extreme values

Syntax

function flmin()

function flmax()

function ffrac()

double precision function dflmin()

double precision function dflmax()

double precision function dffrac()

function inmax()

Description

Functions `flmin` and `flmax` return the minimum and maximum positive floating point values respectively. Functions `dflmin` and `dflmax` return the minimum and maximum positive double precision floating point values. Function `inmax` returns the maximum positive integer value.

The functions `ffrac` and `dffrac` return the fractional accuracy of single and double precision floating point numbers respectively. These are the smallest numbers that can be added to 1.0 without being lost.

These functions can be used by programs that must scale algorithms to the numerical range of the processor.

Files

/usr/lib/libF77.a

VAX **flush (3f)**
Unsupported

Name

flush – flush output to a logical unit

Syntax

subroutine flush (lunit)

Description

The `flush` function causes the contents of the buffer for logical unit *lunit* to be flushed to the associated file. This is most useful for logical units 0 and 6 when they are both associated with the control terminal.

Files

/usr/lib/libI77.a

See Also

fclose(3s)

Name

fork – create a copy of this process

Syntax

integer function fork()

Description

The `fork` function creates a copy of the calling process. The only distinction between the two processes is that the value returned to one of them (referred to as the 'parent' process) will be the process ID of the copy. The copy is usually referred to as the 'child' process. The value returned to the 'child' process will be zero.

All logical units open for writing are flushed before the fork to avoid duplication of the contents of I/O buffers in the external file(s).

If the returned value is negative, it indicates an error and will be the negation of the system error code. See `perror`(3f).

A corresponding `exec` routine has not been provided because there is no satisfactory way to retain open logical units across the exec. However, the usual function of `fork/exec` can be performed using `system`(3f).

Files

/usr/lib/libU77.a

See Also

fork(2), kill(3f), perror(3f), system(3f), wait(3f)

VAX **fseek(3f)**
Unsupported

Name

fseek, ftell – reposition a file on a logical unit

Syntax

integer function fseek (lunit, offset, from)
integer offset, from

integer function ftell (lunit)

Description

The *lunit* argument must refer to an open logical unit. The *offset* argument is an offset in bytes relative to the position specified by *from*. Valid values for *from* are:

0 meaning 'beginning of the file'
1 meaning 'the current position'
2 meaning 'the end of the file'

The value returned by `fseek` will be 0 if successful, a system error code otherwise. See `perror`(3f).

The `ftell` function returns the current position of the file associated with the specified logical unit. The value is an offset, in bytes, from the beginning of the file. If the value returned is negative, it indicates an error and will be the negation of the system error code. See `perror`(3f).

Files

/usr/lib/libU77.a

See Also

perror(3f), fseek(3s)

Name

getarg, iargc – return command line arguments

Syntax

subroutine getarg (k, arg)
character*(*) arg

function iargc ()

Description

A call to `getarg` will return the **k***th* command line argument in character string *arg*. The 0*th* argument is the command name.

An `iargc` call returns the index of the last command line argument.

Files

/usr/lib/libU77.a

See Also

execve(2), getenv(3f)

VAX **getc (3f)**
Unsupported

Name

getc, fgetc – get a character from a logical unit

Syntax

integer function getc (char)
character char

integer function fgetc (lunit, char)
character char

Description

These routines return the next character from a file associated with a FORTRAN logical unit, bypassing normal FORTRAN I/O. The `getc` function reads from logical unit 5, usually connected to the control terminal input.

The value of each function is a system status code. Zero indicates no error occurred on the read. A –1 indicates end of file was detected. A positive value will be either an ULTRIX system error code or an `f77` I/O error code. For further information, see `perror`(3f).

Files

/usr/lib/libU77.a

See Also

intro(2), getc(3s), perror(3f)

Name

getcwd – get pathname of current working directory

Syntax

integer function getcwd (dirname)
character*(*) dirname

Description

The pathname of the default directory for creating and locating files will be returned in *dirname*. The value of the function will be zero if successful; an error code otherwise.

Restrictions

Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.

Files

/usr/lib/libU77.a

See Also

chdir(3f), perror(3f)

Name

getenv – get value of environment variables

Syntax

subroutine getenv (ename, evalue)
character*(*) ename, evalue

Description

The `getenv` subroutine searches the environment list for a string of the form *ename=value* and returns *value* in *evalue* if such a string is present, otherwise fills *evalue* with blanks. For further information, see `environ`(7).

Files

/usr/lib/libU77.a

See Also

execve(2), environ(7)

Name

getlog – get user's login name

Syntax

subroutine getlog (name)
character*(*) name

character*(*) function getlog()

Description

The `getlog` subroutine will return the user's login name or all blanks if the process is running detached from a terminal.

Files

/usr/lib/libU77.a

See Also

getlogin(3)

VAX **getpid(3f)**
Unsupported

Name

getpid – get process ID

Syntax

integer function getpid()

Description

The `getpid` subroutine returns the process ID number of the current process.

Files

/usr/lib/libU77.a

See Also

getpid(2)

Name

getuid, getgid – get user or group ID of the caller

Syntax

integer function getuid()

integer function getgid()

Description

These functions return the real user or group ID of the user of the process.

Files

/usr/lib/libU77.a

See Also

getuid(2)

VAX **hostnm(3f)**
Unsupported

Name

hostnm – get name of current host

Syntax

integer function hostnm (name)
character*(*) name

Description

This function puts the name of the current host into character string *name*.

Return Value

The return value should be 0; any other value indicates an error.

Files

/usr/lib/libU77.a

See Also

gethostname(2)

Name

idate, itime – return date or time in numerical form

Syntax

subroutine idate (iarray)
integer iarray(3)

subroutine itime (iarray)
integer iarray(3)

Description

The subroutine `idate` returns the current date in *iarray*. The order is: day, mon, year. Month will be in the range 1-12. Year will be ≥ 1969.

The `itime` routine returns the current time in *iarray*. The order is: hour, minute, second.

Files

/usr/lib/libU77.a

See Also

ctime(3), fdate(3f)

VAX **index(3f)**
Unsupported

Name

index, rindex, lnblnk, len – tell about character objects

Syntax

(intrinsic) function index (string, substr)
character*(*) string, substr

integer function rindex (string, substr)
character*(*) string, substr

function lnblnk (string)
character*(*) string

(intrinsic) function len (string)
character*(*) string

Description

The `index` and `rindex` subroutines return the index of the first (last) occurrence of the substring *substr* in *string,* or zero if it does not occur. The `index` subroutine is an f77 intrinsic function. The `rindex` subroutine is a library routine.

The `lnblnk` subroutine returns the index of the last non-blank character in *string*. This is useful since all f77 character objects are fixed length, blank padded. Intrinsic function *len* returns the size of the character object argument.

Files

/usr/lib/libF77.a

Name

ioinit – change f77 I/O initialization

Syntax

logical function ioinit (cctl, bzro, apnd, prefix, vrbose)
logical cctl, bzro, apnd, vrbose
character*(*) prefix

Description

This routine will initialize several global parameters in the f77 I/O system, and attach externally defined files to logical units at run time. The effect of the flag arguments applies to logical units opened after `ioinit` is called. The exception is the preassigned units, 5 and 6, to which *cctl* and *bzro* will apply at any time. The `ioinit` routine is written in FORTRAN-77.

By default, carriage control is not recognized on any logical unit. If *cctl* is **.true.** then carriage control will be recognized on formatted output to all logical units except unit 0, the diagnostic channel. Otherwise the default will be restored.

By default, trailing and embedded blanks in input data fields are ignored. If *bzro* is **.true.** then such blanks will be treated as zeros. Otherwise the default will be restored.

By default, all files opened for sequential access are positioned at their beginning. It is sometimes necessary or convenient to open at the END-OF-FILE so that a write will append to the existing data. If *apnd* is **.true.** then files opened subsequently on any logical unit will be positioned at their end upon opening. A value of **.false.** will restore the default behavior.

The `ioinit` routine may be used to associate file names with FORTRAN logical unit numbers through environment variables. See the "Introduction to the f77 I/O Library" in the *ULTRIX Supplementary Documents* for a more general way of doing this. If the argument *prefix* is a non-blank string, then names of the form **prefix**NN will be sought in the program environment. The value associated with each such name found will be used to open logical unit NN for formatted sequential access. For example, if f77 program *myprogram* included the call:

```
call ioinit (.true., .false., .false., FORT, .false.)
```

then the following sequence:

```
% setenv FORT01 mydata
% setenv FORT12 myresults
% myprogram
```

would result in logical unit 1 opened to file *mydata* and logical unit 12 opened to file *myresults* . Both files would be positioned at their beginning. Any formatted output would have column 1 removed and interpreted as carriage control. Embedded and trailing blanks would be ignored on input.

If the argument *vrbose* is **.true.** then `ioinit` will report on its activity.

The effect of:

```
call ioinit (.true., .true., .false., , .false.)
```

can be achieved without the actual call by including "–lI66" on the `f77` command line. This gives carriage control on all logical units except 0, causes files to be opened at their beginning, and causes blanks to be interpreted as zero's.

The internal flags are stored in a labeled common block with the following definition:

```
integer*2 ieof, ictl, ibzr
common /ioiflg/ ieof, ictl, ibzr
```

Restrictions

The *prefix* argument can be no longer than 30 characters. A pathname associated with an environment name can be no longer than 255 characters.

The "+" carriage control does not work.

Files

/usr/lib/libI77.a	f77 I/O library
/usr/lib/libI66.a	Sets older FORTRAN I/O modes

See Also

getarg(3f), getenv(3f)
"Introduction to the f77 I/O Library," *ULTRIX Supplementary Documents,* Vol. II: Programmer

Name

kill – send a signal to a process

Syntax

function kill (pid, signum)
integer pid, signum

Description

The *pid* argument must be the process id of one of the user processes. The *signum* argument must be a valid signal number. For further information, see `sigvec`(2).

Return Value

The returned value will be 0 if successful and an error code otherwise.

Files

/usr/lib/libU77.a

See Also

kill(2), sigvec(2), signal(3f), fork(3f), perror(3f)

Name

link, symlink – make a link to an existing file

Syntax

function link (name1, name2)
character*(*) name1, name2

integer function symlnk (name1, name2)
character*(*) name1, name2

Description

The *name1* must be the pathname of an existing file. The *name2* is a pathname to be linked to file *name1*. The *name2* must not already exist.

The `symlnk` subroutine creates a symbolic link to *name1*.

Restrictions

Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.

Return Value

The returned value will be 0 if successful and a system error code otherwise.

Files

/usr/lib/libU77.a

See Also

link(2), symlink(2), perror(3f), unlink(3f)

Name

loc – return the address of an object

Syntax

function loc (arg)

Description

The returned value will be the address of *arg*.

Files

/usr/lib/libU77.a

VAX **long(3f)**
Unsupported

Name

long, short – integer object conversion

Syntax

integer*4 function long (int2)
integer*2 int2

integer*2 function short (int4)
integer*4 int4

Description

These functions provide conversion between short and long integer objects. The `long` subroutine is useful when constants are used in calls to library routines and the code is to be compiled with "-i2". The `short` subroutine is useful in similar context when an otherwise long object must be passed as a short integer.

Files

/usr/lib/libF77.a

Name

perror, gerror, ierrno – get system error messages

Syntax

subroutine perror (string)
character*(*) string

subroutine gerror (string)
character*(*) string

character*(*) function gerror()

function ierrno()

Description

The `perror` subroutine will write a message to FORTRAN logical unit 0 appropriate to the last detected system error. The *string* will be written preceding the standard error message.

The `gerror` subroutine returns the system error message in character variable *string*. The `gerror` subroutine may be called either as a subroutine or as a function.

The `ierrno` subroutine will return the error number of the last detected system error. This number is updated only when an error actually occurs. Most routines and I/O statements that might generate such errors return an error code after the call. That value is a more reliable indicator of what caused the error condition.

Error Codes

ULTRIX system error codes are described in `intro`(2). The f77 I/O error codes and their meanings are:

```
100    ''error in format''
101    ''illegal unit number''
102    ''formatted io not allowed''
103    ''unformatted io not allowed''
104    ''direct io not allowed''
105    ''sequential io not allowed''
106    ''can't backspace file''
107    ''off beginning of record''
108    ''can't stat file''
109    ''no * after repeat count''
110    ''off end of record''
111    ''truncation failed''
112    ''incomprehensible list input''
113    ''out of free space''
114    ''unit not connected''
115    ''invalid data for integer format term''
116    ''invalid data for logical format term''
117    '''new' file exists''
118    ''can't find 'old' file''
119    ''opening too many files or unknown system error''
120    ''requires seek ability''
121    ''illegal argument''
```

```
122     ''negative repeat count''
123     ''illegal operation for unit''
124     ''invalid data for d, e., f, or g format term''
```

Restrictions

The *string* in the call to `perror` can be no longer than 127 characters.

The length of the string returned by `gerror` is determined by the calling program.

Files

/usr/lib/libU77.a

See Also

intro(2), perror(3)
''Introduction to the f77 I/O Library,'' *ULTRIX Supplementary Documents,* Vol. II: Programmer

Name

putc, fputc – write a character to a FORTRAN logical unit

Syntax

integer function putc (char)
character char

integer function fputc (lunit, char)
character char

Description

These functions write a character to the file associated with a FORTRAN logical unit bypassing normal FORTRAN I/O. The `putc` routine writes to logical unit 6, normally connected to the control terminal output.

The value of each function will be zero unless some error occurred; a system error code otherwise. For further information, see `perror`(3f).

Files

/usr/lib/libU77.a

See Also

intro(2), perror(3f), putc(3s)

VAX **qsort(3f)**
Unsupported

Name

qsort – quick sort

Syntax

subroutine qsort (array, len, isize, compar)
external compar
integer*2 compar

Description

One dimensional *array* contains the elements to be sorted. The *len* is the number of elements in the array. The *isize* is the size of an element, typically:

4 for **integer** and **real**
8 for **double precision** or **complex**
16 for **double complex**
(length of character object) for **character** arrays

The *compar* is the name of a user supplied integer*2 function that will determine the sorting order. This function will be called with two arguments that will be elements of *array*. The function must return:

negative if arg 1 is considered to precede arg 2
zero if arg 1 is equivalent to arg 2
positive if arg 1 is considered to follow arg 2

On return, the elements of *array* will be sorted.

Files

/usr/lib/libU77.a

See Also

qsort(3)

Name

rand, drand, irand – return random values

Syntax

function irand (iflag)

function rand (iflag)

double precision function drand (iflag)

Description

These functions use `rand`(3) to generate sequences of random numbers. If *iflag* is '1', the generator is restarted and the first random value is returned. If *iflag* is otherwise nonzero, it is used as a new seed for the random number generator, and the first new random value is returned.

Restrictions

The algorithm returns a 31-bit quantity on the VAX.

Return Value

The `irand` subroutine returns positive integers in the range 0 through 2147483647. The `rand` and `drand` subroutines return values in the range 0. through 1.0.

Files

/usr/lib/libF77.a

See Also

rand(3)

Name

rename – rename a file

Syntax

integer function rename (from, to)
character*(*) from, to

Description

The `rename` subroutine renames a file. The *from* argument must be the pathname of an existing file, and *to* will become the new pathname for the file. If *to* exists, then both *from* and *to* must be the same type of file, and must reside on the same filesystem. If *to* exists, it will be removed first.

Restrictions

Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.

Return Value

The returned value will be 0 if successful and a system error code otherwise.

Files

/usr/lib/libU77.a

See Also

rename(2), perror(3f)

Name

signal – change the action for a signal

Syntax

integer function signal(signum, proc, flag)
integer signum, flag
external proc

Description

When a process incurs a signal, the default action is usually to clean up and abort. For further information, see `signal`(3). The user may choose to write an alternative signal handling routine. A call to `signal` is the way this alternate action is specified to the system.

The *signum* is the signal number. For further information, see `signal`(3). If *flag* is negative, then *proc* must be the name of the user signal handling routine. If *flag* is zero or positive, then *proc* is ignored and the value of *flag* is passed to the system as the signal action definition. In particular, this is how previously saved signal actions can be restored. Two possible values for *flag* have specific meanings: 0 means "use the default action" (See Note), 1 means "ignore this signal".

A positive returned value is the previous action definition. A value greater than 1 is the address of a routine that was to have been called on occurrence of the given signal. The returned value can be used in subsequent calls to `signal` in order to restore a previous action definition. A negative returned value is the negation of a system error code. For further information, see `perror`(3f).

NOTE

The `f77` arranges to trap certain signals when a process is started. The only way to restore the default `f77` action is to save the returned value from the first call to `signal`.

If the user signal handler is called, it will be passed the signal number as an integer argument.

Files

/usr/lib/libU77.a

See Also

kill(1), kill(3f), signal(3)

VAX **sleep (3f)**
Unsupported

Name

sleep – suspend execution for an interval

Syntax

subroutine sleep (itime)

Description

The `sleep` subroutine causes the calling process to be suspended for *itime* seconds. The actual time can be up to 1 second less than *itime* due to granularity in system timekeeping.

Files

/usr/lib/libU77.a

See Also

sleep(3)

Name

stat, lstat, fstat – get file status

Syntax

integer function stat (name, statb)
character*(*) name
integer statb(12)

integer function lstat (name, statb)
character*(*) name
integer statb(12)

integer function fstat (lunit, statb)
integer statb(12)

Description

These routines return detailed information about a file. The `stat` and `lstat` routines return information about file *name*. The `fstat` subroutine returns information about the file associated with FORTRAN logical unit *lunit*. The order and meaning of the information returned in array *statb* is as described for the structure `stat` under `stat`(2). The spare values are not included.

Restrictions

Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.

Return Value

The value of either function will be zero if successful; an error code otherwise.

Files

/usr/lib/libU77.a

See Also

stat(2), access(3f), perror(3f), time(3f)

Name

system – execute an ULTRIX command

Syntax

integer function system (string)
character*(*) string

Description

The `system` routine causes *string* to be given to your shell as input as if the string had been typed as a command. If environment variable SHELL is found, its value will be used as the command interpreter (shell); otherwise `sh`(1) is used.

The current process waits until the command terminates. The returned value will be the exit status of the shell. See `wait`(2) for an explanation of this value.

Restrictions

The *string* cannot be longer than NCARGS_R50 characters, as defined in <sys/param.h>.

Files

/usr/lib/libU77.a

See Also

execve(2), wait(2), system(3)

Name

time, ctime, ltime, gmtime – return system time

Syntax

integer function time()

character*(*) function ctime (stime)
integer stime

subroutine ltime (stime, tarray)
integer stime, tarray(9)

subroutine gmtime (stime, tarray)
integer stime, tarray(9)

Description

The `time` routine returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds. This is the value of the ULTRIX system clock.

The `ctime` routine converts a system time to a 24-character ASCII string. The format is described under `ctime(3)`. No 'newline' or NULL will be included.

The `ltime` and `gmtime` routines dissect ULTRIX time into month, day and seconds, either for the local time zone or as GMT. The order and meaning of each element returned in *tarray* is described under `ctime(3)`.

Files

/usr/lib/libU77.a

See Also

ctime(3), fdate(3f), idate(3f)

Name

topen, tclose, tread, twrite, trewin, tskipf, tstate – f77 tape I/O

Syntax

integer function topen (tlu, devnam, label)
integer tlu
character*(*) devnam
logical label

integer function tclose (tlu)
integer tlu

integer function tread (tlu, buffer)
integer tlu
character*(*) buffer

integer function twrite (tlu, buffer)
integer tlu
character*(*) buffer

integer function trewin (tlu)
integer tlu

integer function tskipf (tlu, nfiles, nrecs)
integer tlu, nfiles, nrecs

integer function tstate (tlu, fileno, recno, errf, eoff, eotf, tcsr)
integer tlu, fileno, recno, tcsr
logical errf, eoff, eotf

Description

These functions provide a simple interface between f77 and magnetic tape devices. A ''tape logical unit,'' *tlu*, is ''topen''ed in much the same way as a normal f77 logical unit is ''open''ed. All other operations are performed via the *tlu*. The *tlu* has no relationship at all to any normal f77 logical unit.

It should be noted that these functions return integers and should be declared in the user program before using them. The default function return value for the f77 compiler is float. The failure to declare these functions will produce improper results.

The `topen` function associates a device name with a *tlu*. The *tlu* must be in the range 0 to 3. The logical argument *label* should indicate whether the tape includes a tape label. This is used by `trewin` below. The `topen` function does not move the tape. The normal returned value is 0. If the value of the function is negative, an error has occurred. See `perror`(3f) for details.

The `tclose` function closes the tape device channel and removes its association with *tlu*. The normal returned value is 0. A negative value indicates an error.

The `tread` function reads the next physical record from tape to *buffer*. The *buffer* must be of type **character**. The size of *buffer* should be large enough to hold the largest physical record to be read. The actual number of bytes read will be returned as the value of the function. If the value is 0, the end-of-file has been detected. A negative value indicates an error.

The `twrite` function writes a physical record to tape from *buffer*. The physical record length will be the size of *buffer*. The *buffer* **must** be of type **character**. The number of bytes written will be returned. A value of 0 or negative indicates an error.

The `trewin` function rewinds the tape associated with *tlu* to the beginning of the first data file. If the tape is a labeled tape then the label is skipped over after rewinding. For further information, see `topen` above. The normal returned value is 0. A negative value indicates an error.

The `tskipf` function allows the user to skip over files and/or records. First, *nfiles* end-of-file marks are skipped. If the current file is at EOF, this counts as 1 file to skip. (Note: This is the way to reset the EOF status for a *tlu*.) Next, *nrecs* physical records are skipped over. The normal returned value is 0. A negative value indicates an error.

Finally, the `tstate` function allows the user to determine the logical state of the tape I/O channel and to see the tape drive control status register. The values of *fileno* and *recno* will be returned and indicate the current file and record number. The logical values *errf*, *eoff*, and *eotf* indicate an error has occurred, the current file is at EOF, or the tape has reached logical end-of-tape. End-of-tape (EOT) is indicated by an empty file, often referred to as a double EOF mark. It is not allowed to read past EOT although it is allowed to write. The value of *tcsr* will reflect the tape drive control status register. For further details, see `tu`(4).

Files

/usr/lib/libU77.a

See Also

fseek(3f), perror(3f), tu(4)

VAX **traper(3f)**
Unsupported

Name

traper – trap arithmetic errors

Syntax

integer function traper (mask)

Description

Integer overflow and floating point underflow are not normally trapped during execution. This routine enables these traps by setting status bits in the process status word. These bits are reset on entry to a subprogram, and the previous state is restored on return. Therefore, this routine must be called *inside* each subprogram in which these conditions should be trapped. If the condition occurs and trapping is enabled, signal SIGFPE is sent to the process. For further information, see `signal`(3).

The argument has the following meaning:

value	meaning
0	do not trap either condition
1	trap integer overflow only
2	trap floating underflow only
3	trap both the above

The previous value of these bits is returned.

Files

/usr/lib/libF77.a

See Also

signal(3), signal(3f)

Name

trapov – trap and repair floating point overflow

Syntax

subroutine trapov (numesg, rtnval)
double precision rtnval

Description

NOTE: This routine applies only to the older VAX 11/780's. VAX computers made or upgraded since spring 1983 (REV 7) handle errors differently. See trpfpe(3f) for the newer error handler. This routine has always been ineffective on the VAX 11/750. It is a null routine on the PDP11.

This call sets up signal handlers to trap arithmetic exceptions and the use of illegal operands. Trapping arithmetic exceptions allows the user's program to proceed from instances of floating point overflow or divide by zero. The result of such operations will be an illegal floating point value. The subsequent use of the illegal operand will be trapped and the operand replaced by the specified value.

The first *numesg* occurrences of a floating point arithmetic error will cause a message to be written to the standard error file. If the resulting value is used, the value given for *rtnval* will replace the illegal operand generated by the arithmetic error. The *rtnval* must be a double precision value. For example, "0d0" or "dflmax()".

Restrictions

Other arithmetic exceptions can be trapped but not repaired.

There is no way to distinguish between an integer value of 32768 and the illegal floating point form. Therefore such an integer value may get replaced while repairing the use of an illegal operand.

Files

/usr/lib/libF77.a

See Also

signal(3f), trpfpe(3f)

VAX **trpfpe(3f)**
Unsupported

Name

trpfpe, fpecnt – trap and repair floating point faults

Syntax

subroutine trpfpe (numesg, rtnval)
double precision rtnval

integer function fpecnt ()

common /fpeflt/ fperr
logical fperr

Description

The `trpfpe` routine sets up a signal handler to trap arithmetic exceptions. If the exception is due to a floating point arithmetic fault, the result of the operation is replaced with the *rtnval* specified. The *rtnval* must be a double precision value. For example, ''0d0'' or ''dflmax()''.

The first *numesg* occurrences of a floating point arithmetic error will cause a message to be written to the standard error file. Any exception that cannot be repaired will result in the default action, typically an abort with core image.

The `fpecnt` routine returns the number of faults since the last call to `trpfpe`.

The logical value in the common block labeled */peflt* will be set to .true. each time a fault occurs.

Restrictions

This routine works only for *faults*, not *traps*. This is primarily due to the VAX architecture.

If the operation involves changing the stack pointer, it cannot be repaired. This seldom should be a problem with the f77 compiler, but such an operation might be produced by the optimizer.

The POLY and EMOD opcodes are not dealt with.

Files

/usr/lib/libF77.a

See Also

signal(3f)

Name

ttynam, isatty – find name of a terminal port

Syntax

character*(*) function ttynam (lunit)

logical function isatty (lunit)

Description

The `ttynam` subroutine returns a blank padded pathname of the terminal device associated with logical unit *lunit*.

The `isatty` subroutine returns .true. Otherwise, if *lunit* is associated with a terminal device, .false.

Diagnostics

The `ttynam` returns an empty string (all blanks) if *lunit* is not associated with a terminal device in directory `/dev`.

Files

/dev/*
/usr/lib/libU77.a

VAX **unlink(3f)**
Unsupported

Name

unlink – remove a directory entry

Syntax

integer function unlink (name)
character*(*) name

Description

The `unlink` subroutine causes the directory entry specified by pathname *name* to be removed. If this was the last link to the file, the contents of the file are lost.

Restrictions

Pathnames can be no longer than MAXPATHLEN as defined in <sys/param.h>.

Return Value

The returned value will be zero if successful; a system error code otherwise.

Files

/usr/lib/libU77.a

See Also

unlink(2), link(3f), perror(3f)

Name

wait – wait for a process to terminate

Syntax

integer function wait (status)
integer status

Description

The `wait` routine causes its caller to be suspended until a signal is received or one of its child processes terminates. If any child has terminated since the last `wait`, return is immediate; if there are no children, return is immediate with an error code.

If the returned value is positive, it is the process ID of the child and *status* is its termination status. For further information, see `wait`(2). If the returned value is negative, it is the negation of a system error code.

Files

/usr/lib/libU77.a

See Also

wait(2), kill(3f), perror(3f), signal(3f)

Name

lib2648 – subroutines for the HP 2648 graphics terminal

Syntax

#include <stdio.h>

typedef char *bitmat;
FILE *trace;

cc file.c **–l2648**

Description

Lib2648 is a general purpose library of subroutines useful for interactive graphics on the Hewlett-Packard 2648 graphics terminal. To use it you must call the routine *ttyinit*() at the beginning of execution, and *done*() at the end of execution. All terminal input and output must go through the routines *rawchar*, *readline*, *outchar*, and *outstr*.

Lib2648 does the necessary ^E/^F handshaking if *getenv(''TERM'')* returns ''hp2648'', as it will if set by *tset*(1). Any other value, including for example ''2648'', will disable handshaking.

Bit matrix routines are provided to model the graphics memory of the 2648. These routines are generally useful, but are specifically useful for the *update* function which efficiently changes what is on the screen to what is supposed to be on the screen. The primative bit matrix routines are *newmat*, *mat*, and *setmat*.

The file *trace*, if non-null, is expected to be a file descriptor as returned by *fopen*. If so, *lib2648* will trace the progress of the output by writing onto this file. It is provided to make debugging output feasible for graphics programs without messing up the screen or the escape sequences being sent. Typical use of trace will include:

```
switch (argv[1][1]) {
case 'T':
        trace = fopen("trace", "w");
        break;
...
if (trace)
        fprintf(trace, "x is %d, y is %s\n", x, y);
...
dumpmat("before update", xmat);
```

Routines

agoto(x, y)
Move the alphanumeric cursor to position (x, y), measured from the upper left corner of the screen.

aoff() Turn the alphanumeric display off.

aon() Turn the alphanumeric display on.

Unsupported

areaclear(rmin, cmin, rmax, cmax)
Clear the area on the graphics screen bordered by the four arguments. In normal mode the area is set to all black, in inverse video mode it is set to all white.

beep() Ring the bell on the terminal.

bitcopy(dest, src, rows, cols) bitmat dest,
Copy a *rows* by *cols* bit matrix from *src* to (user provided) *dest.*

cleara() Clear the alphanumeric display.

clearg() Clear the graphics display. Note that the 2648 will only clear the part of the screen that is visible if zoomed in.

curoff() Turn the graphics cursor off.

curon() Turn the graphics cursor on.

dispmsg(str, x, y, maxlen) char *str;
Display the message *str* in graphics text at position *(x, y)*. The maximum message length is given by *maxlen*, and is needed to for dispmsg to know how big an area to clear before drawing the message. The lower left corner of the first character is at *(x, y)*.

done() Should be called before the program exits. Restores the tty to normal, turns off graphics screen, turns on alphanumeric screen, flushes the standard output, etc.

draw(x, y)
Draw a line from the pen location to *(x, y)*. As with all graphics coordinates, *(x, y)* is measured from the bottom left corner of the screen. *(x, y)* coordinates represent the first quadrant of the usual Cartesian system.

drawbox(r, c, color, rows, cols)
Draw a rectangular box on the graphics screen. The lower left corner is at location *(r, c)*. The box is *rows* rows high and *cols* columns wide. The box is drawn if *color* is 1, erased if *color* is 0. *(r, c)* absolute coordinates represent row and column on the screen, with the origin at the lower left. They are equivalent to *(x, y)* except for being reversed in order.

dumpmat(msg, m, rows, cols) char *msg; bitmat m;
If *trace* is non-null, write a readable ASCII representation of the matrix *m* on *trace*. *Msg* is a label to identify the output.

emptyrow(m, rows, cols, r) bitmat m;
Returns 1 if row *r* of matrix *m* is all zero, else returns 0. This routine is provided because it can be implemented more efficiently with a knowledge of the internal representation than a series of calls to *mat.*

error(msg) char *msg;
Default error handler. Calls *message(msg)* and returns. This is called by certain routines in *lib2648*. It is also suitable for calling by the user program. It is probably a good idea for a fancy graphics program to supply its own error procedure which uses *setjmp*(3) to restart the program.

gdefault()
Set the terminal to the default graphics modes.

goff() Turn the graphics display off.

gon() Turn the graphics display on.

koff() Turn the keypad off.

kon() Turn the keypad on. This means that most special keys on the terminal (such as the alphanumeric arrow keys) will transmit an escape sequence instead of doing their function locally.

line(x1, y1, x2, y2)
Draw a line in the current mode from *(x1, y1)* to *(x2, y2)*. This is equivalent to *move(x1, y1); draw(x2, y2);* except that a bug in the terminal involving repeated lines from the same point is compensated for.

lowleft() Move the alphanumeric cursor to the lower left (home down) position.

mat(m, rows, cols, r, c) bitmat m;
Used to retrieve an element from a bit matrix. Returns 1 or 0 as the value of the *[r, c]* element of the *rows* by *cols* matrix *m*. Bit matrices are numbered *(r, c)* from the upper left corner of the matrix, beginning at (0, 0). *R* represents the row, and *c* represents the column.

message(str) char *str;
Display the text message *str* at the bottom of the graphics screen.

minmax(g, rows, cols, rmin, cmin, rmax, cmax) bitmat g;
int *rmin, *cmin, *rmax, *cmax;
Find the smallest rectangle that contains all the 1 (on) elements in the bit matrix g. The coordinates are returned in the variables pointed to by rmin, cmin, rmax, cmax.

move(x, y)
Move the pen to location *(x, y)*. Such motion is internal and will not cause output until a subsequent *sync()*.

movecurs(x, y)
Move the graphics cursor to location *(x, y)*.

bitmat newmat(rows, cols)
Create (with *malloc*(3)) a new bit matrix of size *rows* by *cols*. The value created (e.g. a pointer to the first location) is returned. A bit matrix can be freed directly with *free*.

outchar(c) char c;
Print the character *c* on the standard output. All output to the terminal should go through this routine or *outstr*.

outstr(str) char *str;
Print the string str on the standard output by repeated calls to *outchar*.

printg() Print the graphics display on the printer. The printer must be configured as device 6 (the default) on the HPIB.

char rawchar()
Read one character from the terminal and return it. This routine or *readline* should be used to get all input, rather than *getchar*(3).

rboff() Turn the rubber band line off.

rbon() Turn the rubber band line on.

char *rdchar(c) char c;
Return a readable representation of the character *c*. If *c* is a printing character it returns itself, if a control character it is shown in the ^X notation, if negative an apostrophe is prepended. Space returns ^`, rubout returns ^?.

NOTE: A pointer to a static place is returned. For this reason, it will not work to pass rdchar twice to the same *fprintf/sprintf* call. You must instead save one of the values in your own buffer with strcpy.

readline(prompt, msg, maxlen) char *prompt, *msg;
Display *prompt* on the bottom line of the graphics display and read one line of text from the user, terminated by a newline. The line is placed in the buffer *msg*, which has size *maxlen* characters. Backspace processing is supported.

setclear() Set the display to draw lines in erase mode. (This is reversed by inverse video mode.)

setmat(m, rows, cols, r, c, val) bitmat m;
The basic operation to store a value in an element of a bit matrix. The *[r, c]* element of *m* is set to *val,* which should be either 0 or 1.

setset() Set the display to draw lines in normal (solid) mode. (This is reversed by inverse video mode.)

setxor() Set the display to draw lines in exclusive or mode.

sync() Force all accumulated output to be displayed on the screen. This should be followed by fflush(stdout). The cursor is not affected by this function. Note that it is normally never necessary to call *sync*, since *rawchar* and *readline* call *sync()* and *fflush(stdout)* automatically.

togvid() Toggle the state of video. If in normal mode, go into inverse video mode, and vice versa. The screen is reversed as well as the internal state of the library.

ttyinit() Set up the terminal for processing. This routine should be called at the beginning of execution. It places the terminal in CBREAK mode, turns off echo, sets the proper modes in the terminal, and initializes the library.

update(mold, mnew, rows, cols, baser, basec) bitmat mold, mnew;
Make whatever changes are needed to make a window on the screen look like *mnew*. *Mold* is what the window on the screen currently looks like. The window has size *rows* by *cols*, and the lower left corner on the screen of the window is *[baser, basec]*. Note: *update* was not intended to be used for the entire screen. It would work but be very slow and take 64K bytes of memory just for mold and mnew. It was intended for 100 by 100 windows with objects in the center of them, and is quite fast for such windows.

vidinv() Set inverse video mode.

vidnorm()
Set normal video mode.

zermat(m, rows, cols) bitmat m;
Set the bit matrix *m* to all zeros.

zoomn(size)
Set the hardware zoom to value *size*, which can range from 1 to 15.

zoomoff() Turn zoom off. This forces the screen to zoom level 1 without affecting the current internal zoom number.

zoomon() Turn zoom on. This restores the screen to the previously specified zoom size.

Diagnostics

The routine *error* is called when an error is detected. The only error currently detected is overflow of the buffer provided to *readline*.

Subscripts out of bounds to *setmat* return without setting anything.

Files

/usr/lib/lib2648.a

See Also

fed(1)

Name

acc – ACC LH/DH IMP interface

Syntax

pseudo-device imp
device acc0 at uba0 csr 0167600 vector accrint accxint

Description

The `acc` device provides a Local Host/Distant Host interface to an IMP. It is normally used when participating in the DARPA Internet. The controller itself is not accessible to users, but instead provides the hardware support to the IMP interface described in `imp`(4). When configuring, the `imp` pseudo-device must also be included.

Diagnostics

acc%d: not alive.
The initialization routine was entered even though the device did not autoconfigure. This indicates a system problem.

acc%d: can't initialize.
Insufficient UNIBUS resources existed to initialize the device. This is likely to occur when the device is run on a buffered data path on an 11/750 and other network interfaces are also configured to use buffered data paths, or when it is configured to use buffered data paths on an 11/730 (which has none).

acc%d: imp doesn't respond, icsr=%b.
The driver attempted to initialize the device, but the IMP failed to respond after 500 tries. Check the cabling.

acc%d: stray xmit interrupt, csr=%b.
An interrupt occurred when no output had previously been started.

acc%d: output error, ocsr=%b, icsr=%b.
The device indicated a problem sending data on output.

acc%d: input error, csr=%b.
The device indicated a problem receiving data on input.

acc%d: bad length=%d.
An input operation resulted in a data transfer of less than 0 or more than 1008 bytes of data into memory (according to the word count register). This should never happen as the maximum size of a host-IMP message is 1008 bytes.

VAX **ad(4)**
Unsupported

Name

ad – Data Translation A/D converter

Syntax

device ad0 at uba0 csr 0170400 vector adintr

Description

The `ad` converter provides the interface to the Data Translation A/D converter. This is not a real-time driver, but merely allows the user process to sample the board's channels one at a time. Each minor device selects a different A/D board.

The driver communicates to a user process by means of ioctls. The AD_CHAN ioctl selects which channel of the board to read. For example,

```
chan = 5; ioctl(fd, AD_CHAN, &chan);
```

selects channel 5. The AD_READ ioctl actually reads the data and returns it to the user process. An example is

```
ioctl(fd, AD_READ, &data);
```

Files

`/dev/ad`

Name

bk – line discipline for machine-machine communication (obsolete)

Syntax

pseudo-device bk

Description

This line discipline provides a replacement for the old and new tty drivers described in `tty`(4) when high speed output to and especially input from another machine is to be transmitted over a asynchronous communications line. The discipline was designed for use by the Berkeley network. It may be suitable for uploading of data from microprocessors into the system. If you are going to send data over asynchronous communications lines at high speed into the system, you must use this discipline, as the system otherwise may detect high input data rates on terminal lines and disables the lines; in any case the processing of such data when normal terminal mechanisms are involved saturates the system.

The line discipline is enabled by a sequence:

```
#include <sgtty.h>
int ldisc = NETLDISC, fildes; ...
ioctl(fildes, TIOCSETD, &ldisc);
```

A typical application program then reads a sequence of lines from the terminal port, checking header and sequencing information on each line and acknowledging receipt of each line to the sender, who then transmits another line of data. Typically several hundred bytes of data and a smaller amount of control information will be received on each handshake.

The old standard teletype discipline can be restored by doing:

```
ldisc = OTTYDISC;
ioctl(fildes, TIOCSETD, &ldisc);
```

While in networked mode, normal teletype output functions take place. Thus, if an 8 bit output data path is desired, it is necessary to prepare the output line by putting it into RAW mode using `ioctl`(2). This must be done **before** changing the discipline with TIOCSETD, as most `ioctl`(2) calls are disabled while in network line-discipline mode.

When in network mode, input processing is very limited to reduce overhead. Currently the input path is only 7 bits wide, with newline the only recognized character, terminating an input record. Each input record must be read and acknowledged before the next input is read as the system refuses to accept any new data when there is a record in the buffer. The buffer is limited in length, but the system guarantees to always be willing to accept input resulting in 512 data characters and then the terminating newline.

User level programs should provide sequencing and checksums on the information to guarantee accurate data transfer.

VAX **bk(4)**
Unsupported

See Also

tty(4)

Name

css – DEC IMP-11A LH/DH IMP interface

Syntax

pseudo-device imp
device css0 at uba0 csr 0167600 flags 10 vector cssrint cssxint

Description

The `css` device provides a Local Host/Distant Host interface to an IMP. It is normally used when participating in the DARPA Internet. The controller itself is not accessible to users, but instead provides the hardware support to the IMP interface described in `imp`(4). When configuring, the `imp` pseudo-device is also included.

Diagnostics

css%d: not alive.
The initialization routine was entered even though the device did not autoconfigure. This is indicates a system problem.

css%d: can't initialize.
Insufficient UNIBUS resources existed to initialize the device. This is likely to occur when the device is run on a buffered data path on an 11/750 and other network interfaces are also configured to use buffered data paths, or when it is configured to use buffered data paths on an 11/730 (which has none).

css%d: imp doesn't respond, icsr=%b.
The driver attempted to initialize the device, but the IMP failed to respond after 500 tries. Check the cabling.

css%d: stray output interrupt csr=%b.
An interrupt occurred when no output had previously been started.

css%d: output error, ocsr=%b icsr=%b.
The device indicated a problem sending data on output.

css%d: recv error, csr=%b.
The device indicated a problem receiving data on input.

css%d: bad length=%d.
An input operation resulted in a data transfer of less than 0 or more than 1008 bytes of data into memory (according to the word count register). This should never happen as the maximum size of a host-IMP message is 1008 bytes.

Name

ct – phototypesetter interface

Syntax

device ct0 at uba0 csr 0167760 vector ctintr

Description

This provides an interface to a Graphic Systems C/A/T phototypesetter. Bytes written on the file specify font, size, and other control information as well as the characters to be flashed. The coding is not described here.

Only one process may have this file open at a time. It is write-only.

Files

/dev/cat

See Also

troff(1)
Phototypesetter interface specification

Name

dh – DH-11/DM-11 communications multiplexer

Syntax

device dh0 at uba? csr 0160020 flags 0x???? vector dhrint dhxint
device dm0 at uba? csr 0170500 flags 0x???? vector dmintr

Description

A DH-11 provides 16 communication lines; DM-11's may be optionally paired with DH-11's to provide modem control for the lines.

Each line attached to the DH-11 communications multiplexer behaves as described in `tty`(4). Input and output for each line may independently be set to run at any of 16 speeds; see `tty`(4) for the encoding.

Bit *i* of flags may be specified for a `dh` to say that a line is not properly connected, and that the line should be treated as hard-wired with carrier always present. Thus specifying ''flags 0x0004'' in the specification of dh0 would cause line ttyh2 to be treated in this way.

The `dh` driver normally uses input silos and polls for input at each clock tick (10 milliseconds) rather than taking an interrupt on each input character.

Diagnostics

dh%d: NXM
No response from UNIBUS on a dma transfer within a timeout period. This is often followed by a UNIBUS adapter error. This occurs most frequently when the UNIBUS is heavily loaded and when devices which hog the bus (such as rk07's) are present. It is not serious.

dh%d: silo overflow
The character input silo overflowed before it could be serviced. This can happen if a hard error occurs when the CPU is running with elevated priority, as the system will then print a message on the console with interrupts disabled. If the Berknet is running on a `dh` line at high speed (for example, 9600 baud), there is only 1/15th of a second of buffering capacity in the silo, and overrun is possible. This may cause a few input characters to be lost to users and a network packet is likely to be corrupted, but the network will recover. It is not serious.

Files

```
/dev/tty[h-o][0-9a-f]
/dev/ttyd[0-9a-f]
```

See Also

tty(4)

Name

dn – DN-11 autocall unit interface

Syntax

device dn0 at uba? csr 0160020 vector dnintr

Description

The `dn` device provides an interface through a DEC DN-11 (or equivalent such as the Able Quadracall) to an auto-call unit (ACU). To place an outgoing call one forks a sub-process which opens the appropriate call unit file, `/dev/cua?` and writes the phone number on it. The parent process then opens the corresponding modem line `/dev/cul?`. When the connection has been established, the open on the modem line, `/dev/cul?` will return and the process will be connected. A timer is normally used to timeout the opening of the modem line.

The codes for the phone numbers are:

0-9	dial 0-9
*	dial * (':' is a synonym)
#	dial # (';' is a synonym)
–	delay 20 milliseconds
<	end-of-number ('e' is a synonym)
=	delay for a second dial tone ('w' is a synonym)
f	force a hangup of any existing connection

The entire telephone number must be presented in a single `write` system call.

By convention, even numbered call units are for 300 baud modem lines, while odd numbered units are for 1200 baud lines. For example, `/dev/cua0` is associated with a 300 baud modem line, `/dev/cul0`, while `/dev/cua1` is associated with a 1200 baud modem line, `/dev/cul1`. For devices such as the Quadracall which simulate multiple DN-11 units, the minor device indicates which outgoing modem to use.

Diagnostics

Two error numbers are of interest at open time.

[EBUSY] The dialer is in use.

[ENXIO] The device doesn't exist, or there's no power to it.

Files

/dev/cua?	call units
/dev/cul?	associated modem lines

See Also

tip(1c)

Name

ec – 3Com 10 Mb/s Ethernet interface

Syntax

device ec0 at uba0 csr 0161000 vector ecrint eccollide ecxint

Description

The `ec` interface provides access to a 10 Mb/s Ethernet network through a 3com controller.

The hardware has 32 kilobytes of dual-ported memory on the UNIBUS. This memory is used for internal buffering by the board, and the interface code reads the buffer contents directly through the UNIBUS.

The host's Internet address is specified at boot time with an SIOCSIFADDR ioctl. The `ec` interface employs the address resolution protocol described in `arp`(4p) to dynamically map between Internet and Ethernet addresses on the local network.

The interface software implements an exponential backoff algorithm when notified of a collision on the cable. This algorithm utilizes a 16-bit mask and the VAX-11's interval timer in calculating a series of random backoff values. The algorithm is as follows:

1. Initialize the mask to be all 1's.
2. If the mask is zero, 16 retries have been made and we give up.
3. Shift the mask left one bit and formulate a backoff by masking the interval timer with the mask (this is actually the two's complement of the value).
4. Use the value calculated in step 3 to delay before retransmitting the packet. The delay is done in a software busy loop.

The interface normally tries to use a ''trailer'' encapsulation to minimize copying data on input and output. This may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

Diagnostics

ec%d: send error.
After 16 retransmissions using the exponential backoff algorithm described above, the packet was dropped.

ec%d: input error (offset=%d).
The hardware indicated an error in reading a packet off the cable or an illegally sized packet. The buffer offset value is printed for debugging purposes.

ec%d: can't handle af%d.
The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

VAX

ec(4)
Unsupported

See Also

intro(4n), inet(4f), arp(4p)

Name

en – Xerox 3 Mb/s Ethernet interface

Syntax

device en0 at uba0 csr 0161000 vector enrint enxint encollide

Description

The en interface provides access to a 3 Mb/s Ethernet network. Due to limitations in the hardware, DMA transfers to and from the network must take place in the lower 64K bytes of the UNIBUS address space.

The network number is specified with a SIOCSIFADDR ioctl; the host's address is discovered by probing the on-board Ethernet address register. No packets will be sent or accepted until a network number is supplied.

The interface software implements an exponential backoff algorithm when notified of a collision on the cable. This algorithm utilizes a 16-bit mask and the VAX-11's interval timer in calculating a series of random backoff values. The algorithm is as follows:

1. Initialize the mask to be all 1's.
2. If the mask is zero, 16 retries have been made and we give up.
3. Shift the mask left one bit and formulate a backoff by masking the interval timer with the mask (this is actually the two's complement of the value).
4. Use the value calculated in step 3 to delay before retransmitting the packet.

The interface handles both Internet and PUP protocol families, with the interface address maintained in Internet format. PUP addresses are converted to Internet addresses by subsituting PUP network and host values for Internet network and local part values.

The interface normally tries to use a ''trailer'' encapsulation to minimize copying data on input and output. This may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

Diagnostics

en%d: output error.
The hardware indicated an error on the previous transmission.

en%d: send error.
After 16 retransmissions using the exponential backoff algorithm described above, the packet was dropped.

en%d: input error.
The hardware indicated an error in reading a packet off the cable.

en%d: can't handle af%d.
The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

See Also

intro(4n), inet(4f)

Name

hy – Network Systems Hyperchannel interface

Syntax

device hy0 at uba0 csr 0172410 vector hyint

Description

The hy interface provides access to a Network Systems Corporation Hyperchannel Adapter.

The network to which the interface is attached is specified at boot time with an SIOCSIFADDR ioctl. The host's address is discovered by reading the adapter status register. The interface will not transmit or receive packets until the network number is known.

Restrictions

If the adapter does not respond to the status command issued during autoconfigure, the adapter is assumed down. A reboot is required to recognize it.

Diagnostics

hy%d: unit number 0x%x port %d type %x microcode level 0x%x.
Identifies the device during autoconfiguration.

hy%d: can't handle af%d.
The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

hy%d: can't initialize.
The interface was unable to allocate UNIBUS resources. This is usually due to having too many network devices on an 11/750 where there are only 3 buffered data paths.

hy%d: NEX - Non Existent Memory.
Non existent memory error returned from hardware.

hy%d: BAR overflow.
Bus address register overflow error returned from hardware.

hy%d: Power Off bit set, trying to reset.
Adapter has lost power, driver will reset the bit and see if power is still out in the adapter.

hy%d: Power Off Error, network shutdown.
Power was really off in the adapter, network connections are dropped. Software does not shut down the network unless power has been off for a while.

hy%d: RECVD MP > MPSIZE (%d).
A message proper was received that is too big. Probable a driver bug. Shouldn't happen.

hy%d: xmit error – len > hy_olen [%d > %d].
Probable driver error. Shouldn't happen.

hy%d: DRIVER BUG – INVALID STATE %d.
The driver state machine reached a non-existent state. Definite driver bug.

hy%d: watchdog timer expired.
A command in the adapter has taken too long to complete. Driver will abort and retry the command.

hy%d: adapter power restored.
Software was able to reset the power off bit, indicating that the power has been restored.

See Also

intro(4n), inet(4f)

Name

ik – Ikonas frame buffer, graphics device interface

Syntax

device ik0 at uba? csr 0172460 vector ikintr

Description

The `ik` program provides an interface to an Ikonas frame buffer graphics device. Each minor device is a different frame buffer interface board. When the device is opened, its interface registers are mapped, by virtual memory, into the user processes address space. This allows the user process very high bandwidth to the frame buffer with no system call overhead.

Bytes written or read from the device are DMA'd from or to the interface. The frame buffer XY address, its addressing mode, etc. must be set up by the user process before calling write or read.

Other communication with the driver is by ioctls. The IK_GETADDR ioctl returns the virtual address where the user process can find the interface registers. The IK_WAITINT ioctl suspends the user process until the ikonas device has interrupted (for whatever reason — the user process has to set the interrupt enables).

Restrictions

An invalid access (for example, longword) to a mapped interface register can cause the system to crash with a machine check.

Files

`/dev/ik`

VAX

il(4)
Unsupported

Name

il – Interlan 10 Mb/s Ethernet interface

Syntax

device il0 at uba0 csr 0161000 vector ilrint ilcint

Description

The `il` interface provides access to a 10 Mb/s Ethernet network through an Interlan controller.

The host's Internet address is specified at boot time with an SIOCSIFADDR ioctl. The `il` interface employs the address resolution protocol described in `arp`(4p) to dynamically map between Internet and Ethernet addresses on the local network.

The interface normally tries to use a ''trailer'' encapsulation to minimize copying data on input and output. This may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

Diagnostics

il%d: input error.
The hardware indicated an error in reading a packet off the cable or an illegally sized packet.

il%d: can't handle af%d.
The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

See Also

intro(4n), inet(4f), arp(4p)

Name

imp – 1822 network interface

Syntax

pseudo-device imp

Description

The `imp` interface, as described in BBN Report 1822, provides access to an intelligent message processor normally used when participating in the Department of Defense ARPA network. The network interface communicates through a device controller, usually an ACC LH/DH or DEC IMP-11A, with the IMP. The interface is ''reliable'' and ''flow-controlled'' by the host-IMP protocol.

To configure IMP support, one of `acc`(4) and `css`(4) must be included. The network number on which the interface resides is specified at boot time using the SIOCSIFADDR ioctl. The host number is discovered through receipt of NOOP messages from the IMP.

The network interface is always in one of four states: up, down, initializing, or going down. When the system is booted, the interface is marked down. If the hardware controller is successfully probed, the interface enters the initializing state and transmits three NOOP messages to the IMP. It then waits for the IMP to respond with two or more NOOP messages in reply. When it receives these messages it enters the up state. The going down state is entered only when notified by the IMP of an impending shutdown. Packets may be sent through the interface only while it is in the up state. Packets received in any other state are dropped with the error ENETDOWN returned to the caller.

Diagnostics

imp%d: leader error.
The IMP reported an error in a leader (1822 message header). This causes the interface to be reset and any packets queued up for transmission to be purged.

imp%d: going down in 30 seconds.
imp%d: going down for hardware PM.
imp%d: going down for reload software.
imp%d: going down for emergency reset.
The Network Control Center (NCC) is manipulating the IMP. By convention these messages are reported to all hosts on an IMP.

imp%d: reset (host %d/imp %d).
The host has received a NOOP message which caused it to reset its notion of its current address. This normally occurs at boot time, though it may also occur while the system is running (for example, if the IMP-controller cable is disconnected, then reconnected).

imp%d: host dead.
The IMP has noted a host, to which a prior packet was sent, is not up.

imp%d: host unreachable.
The IMP has discovered a host, to which a prior packet was sent, is not accessible.

imp%d: data error.
The IMP noted an error in data transmitted. The host-IMP interface is reset and the host enters the init state (awaiting NOOP messages).

imp%d: interface reset.
The reset process has been completed.

imp%d: marked down.
After receiving a ''going down in 30 seconds'' message, and waiting 30 seconds, the host has marked the IMP unavailable. Before packets may be sent to the IMP again, the IMP must notify the host, through a series of NOOP messages, that it is back up.

imp%d: can't handle af%d.
The interface was handed a message with addresses formatting in an unsuitable address family; the packet was dropped.

See Also

intro(4n), inet(4f), acc(4), css(4)

Name

imp – IMP raw socket interface

Syntax

#include <sys/socket.h>
#include <netinet/in.h>
#include <netimp/if_imp.h>

s = socket(AF_IMPLINK, SOCK_RAW, IMPLINK_IP);

Description

The raw `imp` socket provides direct access to the `imp`(4) network interface. Users send packets through the interface using the `send`(2) calls, and receive packets with the `recv`(2), calls. All outgoing packets must have space for an 1822 96-bit leader on the front. Likewise, packets received by the user will have this leader on the front. The 1822 leader and the legal values for the various fields are defined in the include file *<netimp/if_imp.h>*.

The raw imp interface automatically installs the length and destination address in the 1822 leader of all outgoing packets; these need not be filled in by the user.

Diagnostics

An operation on a socket may fail with one of the following errors:

[EISCONN] when trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected.

[ENOTCONN] when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected.

[ENOBUFS] when the system runs out of memory for an internal data structure.

[EADDRNOTAVAIL] when an attempt is made to create a socket with a network address for which no network interface exists.

See Also

intro(4n), inet(4f), imp(4)

Name

kg – KL-11/DL-11W line clock

Syntax

device kg0 at uba0 csr 0176500 vector kglock

Description

A kl-11 or dl-11w can be used as an alternate real time clock source. When configured, certain system statistics and, optionally, system profiling work will be collected each time the clock interrupts. For optimum accuracy in profiling, the dl-11w should be configured to interrupt at the highest possible priority level. The *kg* device driver automatically calibrates itself to the line clock frequency.

See Also

kgmon(8), config(8)

Name

pcl – DEC CSS PCL-11 B Network Interface

Syntax

device pcl0 at uba? csr 0164200 vector pclxint pclrint

Description

The `pcl` device provides an IP-only interface to the DEC CSS PCL-11 time division multiplexed network bus. The controller itself is not accessible to users.

The hosts's address is specified with the SIOCSIFADDR ioctl. The interface will not transmit or receive any data before its address is defined.

As the PCL-11 hardware is only capable of having 15 interfaces per network, a single-byte host-on-network number is used, with range [1..15] to match the TDM bus addresses of the interfaces.

The interface currently only supports the Internet protocol family and only provides ''natural'' (header) encapsulation.

Diagnostics

pcl%d: can't init.
Insufficient UNIBUS resources existed to initialize the device. This is likely to occur when the device is run on a buffered data path on an 11/750 and other network interfaces are also configured to use buffered data paths, or when it is configured to use buffered data paths on an 11/730 (which has none).

pcl%d: can't handle af%d.
The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

pcl%d: stray xmit interrupt.
An interrupt occured when no output had previously been started.

pcl%d: master.
The TDM bus had no station providing ''bus master'' timing signals, so this interface has assumed the ''master'' role. This message should only appear at most once per UNIBUS INIT on a single system. Unless there is a hardware failure, only one station may be master at at time.

pcl%d: send error, tcr=%b, tsr=%b.
The device indicated a problem sending data on output. If a ''receiver offline'' error is detected, it is not normally logged unless the option PCL_TESTING has been selected, as this causes a lot of console chatter when sending to a down machine. However, this option is quite useful when debugging problems with the PCL interfaces.

pcl%d: rcv error, rcr=%b rsr=%b.
The device indicated a problem receiving data on input.

pcl%d: bad len=%d.
An input operation resulted in a data transfer of less than 0 or more than 1008 bytes of data into memory (according to the word count register). This should never

happen as the maximum size of a PCL message has been agreed upon to be 1008 bytes (same as ArpaNet message).

See Also

intro(4n), inet(4f)

Name

ps – Evans and Sutherland Picture System 2 graphics device interface

Syntax

device ps0 at uba? csr 0172460 vector psintr

Description

The `ps` driver provides access to an Evans and Sutherland Picture System 2 graphics device. Each minor device is a new PS2. When the device is opened, its interface registers are mapped, via virtual memory, into a user process's address space. This allows the user process very high bandwidth to the device with no system call overhead.

DMA to and from the PS2 is not supported. All read and write system calls will fail. All data is moved to and from the PS2 via programmed I/O using the device's interface registers.

Commands are fed to and from the driver using the following ioctls:

PSIOGETADDR
: Returns the virtual address through which the user process can access the device's interface registers.

PSIOAUTOREFRESH
: Start auto refreshing the screen. The argument is an address in user space where the following data resides. The first longword is a *count* of the number of static refresh buffers. The next *count* longwords are the addresses in refresh memory where the refresh buffers lie. The driver will cycle thru these refresh buffers displaying them one by one on the screen.

PSIOAUTOMAP
: Start automatically passing the display file thru the matrix processor and into the refresh buffer. The argument is an address in user memory where the following data resides. The first longword is a *count* of the number of display files to operate on. The next *count* longwords are the address of these display files. The final longword is the address in refresh buffer memory where transformed coordinates are to be placed if the driver is not in double buffer mode (see below).

PSIODOUBLEBUFFER
: Cause the driver to double buffer the output from the map that is going to the refresh buffer. The argument is again a user space address where the real arguments are stored. The first argument is the starting address of refresh memory where the two double buffers are located. The second argument is the length of each double buffer. The refresh mechanism displays the current double buffer, in addition to its static refresh lists, when in double buffer mode.

PSIOSINGLEREFRESH
: Single step the refresh process. That is, the driver does not continually refresh the screen.

PSIOSINGLEMAP
: Single step the matrix process. The driver does not automatically feed display files thru the matrix unit.

PSIOSINGLEBUFFER
Turn off double buffering.

PSIOTIMEREFRESH
The argument is a count of the number of refresh interrupts to take before turning off the screen. This is used to do time exposures.

PSIOWAITREFRESH
Suspend the user process until a refresh interrupt has occurred. If in TIMEREFRESH mode, suspend until count refreshes have occurred.

PSIOSTOPREFRESH
Wait for the next refresh, stop all refreshes, and then return to user process.

PSIOWAITMAP
Wait until a map done interrupt has occurred.

PSIOSTOPMAP
Wait for a map done interrupt, do not restart the map, and then return to the user.

Restrictions

An invalid access (for example, longword) to a mapped interface register can cause the system to crash with a machine check.

Diagnostics

ps device intr.
ps dma intr.
An interrupt was received from the device. This shouldn't happen, check your device configuration for overlapping interrupt vectors.

Files

`/dev/ps`

Name

pup – Xerox PUP-I protocol family

Syntax

#include <sys/types.h>
#include <netpup/pup.h>

Description

The PUP-I protocol family is a collection of protocols layered atop the PUP Level-0 packet format, and utilizing the PUP Internet address format. The PUP family is currently supported only by a raw interface.

Addressing

PUP addresses are composed of network, host, and port portions. The include file *<netpup/pup.h>* defines this address as,

```
struct        pupport {
      u_char  pup_net;
      u_char  pup_host;
      u_char  pup_socket[4];
};
```

Sockets bound to the PUP protocol family utilize the following addressing structure,

```
struct sockaddr_pup {
      short   spup_family;
      short   spup_zero1;
      u_char  spup_net;
      u_char  spup_host;
      u_char  spup_sock[4];
      char    spup_zero2[4];
};
```

Headers

The current PUP support provides only raw access to the 3Mb/s Ethernet. Packets sent through this interface must have space for the following packet header present at the front of the message,

```
struct pup_header {
      u_short pup_length;
      u_char  pup_tcontrol; /* transport control */
      u_char  pup_type;     /* protocol type */
      u_long  pup_id;       /* used by protocols */
      u_char  pup_dnet;     /* destination */
      u_char  pup_dhost;
      u_char  pup_dsock[4];
      u_char  pup_snet;     /* source */
      u_char  pup_shost;
      u_char  pup_ssock[4];
};
```

The sender should fill in the *pup_tcontrol*, *pup_type*, and *pup_id* fields. The remaining fields are filled in by the system. The system checks the message to insure its size is valid and, calulates a checksum for the message. If no checksum should be calculated, the checksum field (the last 16-bit word in the message) should be set to PUP_NOCKSUM.

The *pup_tcontrol* field is restricted to be 0 or PUP_TRACE; PUP_TRACE indicates packet tracing should be performed. The *pup_type* field may not be 0.

On input, the entire packet, including header, is provided the user. No checksum validation is performed.

See Also

intro(4n), pup(4p), en(4)

Name

pup – raw PUP socket interface

Syntax

#include <sys/socket.h>
#include <netpup/pup.h>

socket(AF_PUP, SOCK_RAW, PUPPROTO_BSP);

Description

A raw pup socket provides PUP-I access to an Ethernet network. Users send packets using the `sendto` call, and receive packets with the `recvfrom` call. All outgoing packets must have space present at the front of the packet to allow the PUP header to be filled in. The header format is described in `pup`(4F). Likewise, packets received by the user will have the PUP header on the front. The PUP header and legal values for the various fields are defined in the include file *<netpup/pup.h>*.

The raw pup interface automatically installs the length and source and destination addresses in the PUP header of all outgoing packets; these need not be filled in by the user. The only control bit that may be set in the *tcontrol* field of outgoing packets is the "trace" bit. A checksum is calculated unless the sender sets the checksum field to PUP_NOCKSUM.

Diagnostics

A socket operation may fail and one of the following will be returned:

[EISCONN] When trying to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected.

[ENOTCONN] When trying to send a datagram, but no destination address is specified, and the socket hasn't been connected.

[ENOBUFS] When the system runs out of memory for an internal data structure.

[EADDRNOTAVAIL] When an attempt is made to create a socket with a network address for which no network interface exists.

A `sendto` operation may fail if one of the following is true:

[EINVAL] Insufficient space was left by the user for the PUP header.

[EINVAL] The *pup_type* field was 0 or the *pup_tcontrol* field had a bit other than PUP_TRACE set.

[EMSGSIZE] The message was not an even number of bytes, smaller than MINPUPSIZ, or large than MAXPUPSIZ.

[ENETUNREACH]
The destination address was on a network which was not directly reachable (the raw interface provides no routing support).

See Also

send(2), recv(2), intro(4n), pup(4f)

Name

tm – TM-11/TE-10 magtape interface

Syntax

controller tm0 at uba? csr 0172520 vector tmintr
tape te0 at tm0 drive 0

Description

The tm-11/te-10 combination provides a standard tape drive interface as described in `mtio`(4). Hardware implementing this on the VAX is typified by the Emulex TC-11 controller operating with a Kennedy model 9300 tape transport, providing 800 and 1600 bpi operation at 125 ips.

Restrictions

If any non-data error is encountered on non-raw tape, it refuses to do anything more until closed.

Diagnostics

te%d: no write ring.
An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

te%d: not online.
An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.

te%d: can't switch density in mid-tape.
An attempt was made to write on a tape at a different density than is already recorded on the tape. This message is written on the terminal of the user who tried to switch the density.

te%d: hard error bn%d er=%b.
A tape error occurred at block *bn*; the tm error register is printed in octal with the bits symbolically decoded. Any error is fatal on non-raw tape; when possible the driver will have retried the operation which failed several times before reporting the error.

te%d: lost interrupt.
A tape operation did not complete within a reasonable time, most likely because the tape was taken off-line during rewind or lost vacuum. The controller should, but does not, give an interrupt in these cases. The device will be made available again after this message, but any current open reference to the device will return an error as the operation in progress aborts.

See Also

mt(1), tar(1), tp(1), mtio(4), ht(4), ts(4), mt(4), ut(4)

Name

un – Ungermann-Bass interface

Syntax

device un0 at uba0 csr 0160210 vector unintr

Description

The un interface provides access to a 4 Mb/s baseband network. The hardware uses a standard DEC DR11-W DMA interface in communicating with the host. The Ungermann-Bass hardware incorporates substantial protocol software in the network device in an attempt to offload protocol processing from the host.

The network number on which the interface resides must be specified at boot time with an SIOCSIFADDR ioctl. The host's address is discovered by communicating with the interface. The interface will not transmit or receive any packets before the network number has been defined.

Restrictions

The device does not reset itself properly resulting in the interface getting hung up in a state from which the only recourse is to reboot the system.

Diagnostics

un%d: can't initialize.
Insufficient UNIBUS resources existed for the device to complete initialization. Usually caused by having multiple network interfaces configured using buffered data paths on a data path poor machine such as the 11/750.

un%d: unexpected reset.
The controller indicated a reset when none had been requested. Check the hardware (but see the bugs section below).

un%d: stray interrupt.
An unexpected interrupt was received. The interrupt was ignored.

un%d: input error csr=%b.
The controller indicated an error on moving data from the device to host memory.

un%d: bad packet type %d.
A packet was received with an unknown packet type. The packet is discarded.

un%d: output error csr=%b.
The device indicated an error on moving data from the host to device memory.

un%d: invalid state %d csr=%b.
The driver found itself in an invalid internal state. The state is reset to a base state.

un%d: can't handle af%d.
A request was made to send a message with an address format which the driver does not understand. The message is discarded and an error is returned to the user.

un%d: error limit exceeded.
Too many errors were encountered in normal operation. The driver will attempt to reset the device, desist from attempting any i/o for approximately 60 seconds, then reset itself to a base state in hopes of resyncing itself up with the hardware.

un%d: restarting.
After exceeding its error limit and resetting the device, the driver is restarting operation.

See Also

intro(4n), inet(4f)

VAX **up(4)**
Unsupported

Name

up – unibus storage module controller/drives

Syntax

controller sc0 at uba? csr 0176700 vector upintr
disk up0 at sc0 drive 0

Description

This is a generic UNIBUS storage module disk driver. It is specifically designed to work with the Emulex SC-21 controller. It can be easily adapted to other controllers (although bootstrapping will not necessarily be directly possible.)

Files with minor device numbers 0 through 7 refer to various portions of drive 0; minor devices 8 through 15 refer to drive 1, etc. The standard device names begin with ''up'' followed by the drive number and then a letter a-h for partitions 0-7 respectively. The character ? stands here for a drive number in the range 0-7.

The block files access the disk via the system's normal buffering mechanism and may be read and written without regard to physical disk records. There is also a 'raw' interface which provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation and therefore raw I/O is considerably more efficient when many words are transmitted. The names of the raw files conventionally begin with an extra 'r.'

In raw I/O counts should be a multiple of 512 bytes (a disk sector). Likewise *seek* calls should specify a multiple of 512 bytes.

Disk Support

The driver interrogates the controller's holding register to determine the type of drive attached. The driver recognizes four different drives: AMPEX 9300, CDC 9766, AMPEX Capricorn, and FUJITSU 160. The origin and size of the pseudo-disks on each drive are as follows:

CDC 9766 300M drive partitions:

disk	start	length	cyl
up?a	0	15884	0-26
up?b	16416	33440	27-81
up?c	0	500384	0-822
up?d	341696	15884	562-588
up?e	358112	55936	589-680
up?f	414048	861760	681-822
up?g	341696	158528	562-822
up?h	49856	291346	82-561

AMPEX 9300 300M drive partitions:

disk	start	length	cyl
up?a	0	15884	0-26
up?b	16416	33440	27-81
up?c	0	495520	0-814
up?d	341696	15884	562-588
up?e	358112	55936	589-680
up?f	414048	81312	681-814
up?g	341696	153664	562-814
up?h	49856	291346	82-561

AMPEX Capricorn 330M drive partitions:

disk	start	length	cyl
hp?a	0	15884	0-31
hp?b	16384	33440	32-97
hp?c	0	524288	0-1023
hp?d	342016	15884	668-699
hp?e	358400	55936	700-809
hp?f	414720	109408	810-1023
hp?g	342016	182112	668-1023
hp?h	50176	291346	98-667

FUJITSU 160M drive partitions:

disk	start	length	cyl
up?a	0	15884	0-49
up?b	16000	33440	50-154
up?c	0	263360	0-822
up?d	49600	15884	155-204
up?e	65600	55936	205-379
up?f	121600	141600	380-822
up?g	49600	213600	155-822

It is unwise for all of these files to be present in one installation, since there is overlap in addresses and protection becomes a sticky matter. The up?a partition is normally used for the root file system, the up?b partition as a paging area, and the up?c partition for pack-pack copying (it maps the entire disk). On 160M drives the up?g partition maps the rest of the pack. On other drives both up?g and up?h are used to map the remaining cylinders.

Restrictions

In raw I/O `read` and `write`(2) truncate file offsets to 512-byte block boundaries, and `write` scribbles on the tail of incomplete blocks. Thus, in programs that are likely to access raw devices, `read`, `write` and `lseek`(2) should always deal in 512-byte multiples.

Diagnostics

up%d%c: hard error sn%d cs2=%b er1=%b er2=%b.
An unrecoverable error occurred during transfer of the specified sector in the specified disk partition. The contents of the cs2, er1 and er2 registers are printed in octal and symbolically with bits decoded. The error was either unrecoverable, or a large number of retry attempts (including offset positioning and drive recalibration)

could not recover the error.

up%d: write locked.
The write protect switch was set on the drive when a write was attempted. The write operation is not recoverable.

up%d: not ready.
The drive was spun down or off line when it was accessed. The i/o operation is not recoverable.

up%d: not ready (flakey).
The drive was not ready, but after printing the message about being not ready (which takes a fraction of a second) was ready. The operation is recovered if no further errors occur.

up%d%c: soft ecc sn%d.
A recoverable ECC error occurred on the specified sector of the specified disk partition. This happens normally a few times a week. If it happens more frequently than this the sectors where the errors are occurring should be checked to see if certain cylinders on the pack, spots on the carriage of the drive or heads are indicated.

sc%d: lost interrupt.
A timer watching the controller detecting no interrupt for an extended period while an operation was outstanding. This indicates a hardware or software failure. There is currently a hardware/software problem with spinning down drives while they are being accessed which causes this error to occur. The error causes a UNIBUS reset, and retry of the pending operations. If the controller continues to lose interrupts, this error will recur a few seconds later.

Files

/dev/up[0-7][a-h]	block files
/dev/rup[0-7][a-h]	raw files

See Also

hk(4), hp(4), uda(4)

Name

urx – DEC RX02 floppy disk interface

Syntax

controller fx0 at uba0 csr 0177170 vector rxintr
disk rx0 at fx0 drive 0
disk rx1 at fx0 slave 1

Description

The `urx` device provides access to a DEC RX02 floppy disk unit with M8256 interface module (RX211 configuration). The RX02 uses 8-inch, single-sided, soft-sectored floppy disks (with pre-formatted industry-standard headers) in either single or double density.

Floppy disks handled by the RX02 contain 77 tracks, each with 26 sectors (for a total of 2,002 sectors). The sector size is 128 bytes for single density, 256 bytes for double density. Single density disks are compatible with the RX01 floppy disk unit and with IBM 3740 Series Diskette 1 systems.

In addition to normal ('block' and 'raw') i/o, the driver supports formatting of disks for either density and the ability to invoke a 2 for 1 interleaved sector mapping compatible with the DEC operating system RT-11.

The minor device number is interpreted as follows:

Bit	Description
0	Sector interleaving (1 disables interleaving)
1	Logical sector 1 is on track 1 (0 no, 1 yes)
2	Not used, reserved
Other	Drive number

The two drives in a single RX02 unit are treated as two disks attached to a single controller. Thus, if there are two RX02's on a system, the drives on the first RX02 are ''rx0'' and ''rx1'', while the drives on the second are ''rx2'' and ''rx3''.

When the device is opened, the density of the disk currently in the drive is automatically determined. If there is no floppy in the device, open will fail.

The interleaving parameters are represented in raw device names by the letters 'a' through 'd'. Thus, unit 0, drive 0 is called by one of the following names:

Mapping	Device name	Starting track
interleaved	/dev/rrx0a	0
direct	/dev/rrx0b	0
interleaved	/dev/rrx0c	1
direct	/dev/rrx0d	1

The mapping used on the 'c' device is compatible with the DEC operating system RT-11. The 'b' device accesses the sectors of the disk in strictly sequential order. The 'a' device is the most efficient for disk-to-disk copying.

The I/O requests must start on a sector boundary, involve an integral number of complete sectors, and not go off the end of the disk.

Notes

Even though the storage capacity on a floppy disk is quite small, it is possible to make filesystems on double density disks. For example, the command

```
% mkfs /dev/rx0 1001 13 1 4096 512 32 0 4
```

makes a file system on the double density disk in rx0 with 436 kbytes available for file storage. Using tar(1) gives a more efficient utilization of the available space for file storage. Single density diskettes do not provide sufficient storage capacity to hold file systems.

A number of ioctl(2) calls apply to the rx devices, and have the form

```
#include <vaxuba/rxreg.h>
ioctl(fildes, code, arg)
int *arg;
```

The applicable codes are:

RXIOC_FORMAT	Format the diskette. The density to use is specified by the *arg* argument, 0 gives single density while non-zero gives double density.
RXIOC_GETDENS	Return the density of the diskette (0 or !=0 as above).
RXIOC_WDDMK	On the next write, include a *deleted data address mark* in the header of the first sector.
RXIOC_RDDMK	Return non-zero if the last sector read contained a *deleted data address mark* in its header, otherwise return 0.

Restrictions

A floppy may not be formatted if the header information on sector 1, track 0 has been damaged. Hence, it is not possible to format completely degaussed disks or disks with other formats than the two known by the hardware.

If the drive subsystem is powered down when the machine is booted, the controller won't interrupt.

Diagnostics

The following errors may be returned by the above ioctl calls:

[ENODEV]	Drive not ready; usually because no disk is in the drive or the drive door is open.
[ENXIO]	Nonexistent drive (on open); offset is too large or not on a sector boundary or byte count is not a multiple of the sector size (on read or write); or bad (undefined) ioctl code.
[EIO]	A physical error other than ''not ready'', probably bad media or unknown format.
[EBUSY]	Drive has been opened for exclusive access.
[EBADF]	No write access (on format), or wrong density; the latter can only happen if the disk is changed without closing the device, that is, calling close(2).

rx%d: hard error, trk %d psec %d cs=%b, db=%b, err=%x, %x, %x, %x.
An unrecoverable error was encountered. The track and physical sector numbers, the device registers and the extended error status are displayed.

rx%d: state %d (reset).
The driver entered a bogus state. This should not happen.

Files

```
/dev/rx?
/dev/rrx?[a-d]
```

See Also

tar(1), arff(8v), mkfs(8), newfs(8), rxformat(8v)

Name

ut – UNIBUS TU45 tri-density tape drive interface

Syntax

controller ut0 at uba0 csr 0172440 vector utintr
tape tj0 at ut0 drive 0

Description

The `ut` interface provides access to a standard tape drive interface as describe in `mtio`(4). Hardware implementing this on the VAX is typified by the System Industries SI 9700 tape subsystem. Tapes may be read or written at 800, 1600, and 6250 bpi.

Restrictions

If any non-data error is encountered on non-raw tape, it refuses to do anything more until closed.

Diagnostics

tj%d: no write ring.
An attempt was made to write on the tape drive when no write ring was present; this message is written on the terminal of the user who tried to access the tape.

tj%d: not online.
An attempt was made to access the tape while it was offline; this message is written on the terminal of the user who tried to access the tape.

tj%d: can't change density in mid-tape.
An attempt was made to write on a tape at a different density than is already recorded on the tape. This message is written on the terminal of the user who tried to switch the density.

ut%d: soft error bn%d cs1=%b er=%b cs2=%b ds=%b.
The formatter indicated a corrected error at a density other than 800bpi. The data transferred is assumed to be correct.

ut%d: hard error bn%d cs1=%b er=%b cs2=%b ds=%b.
A tape error occurred at block *bn*. Any error is fatal on non-raw tape; when possible the driver will have retried the operation which failed several times before reporting the error.

tj%d: lost interrupt.
A tape operation did not complete within a reasonable time, most likely because the tape was taken off-line during rewind or lost vacuum. The controller should, but does not, give an interrupt in these cases. The device will be made available again after this message, but any current open reference to the device will return an error as the operation in progress aborts.

See Also

mt(1), mtio(4)

Name

uu – TU58/DECtape II UNIBUS cassette interface

Syntax

options UUDMA
device uu0 at uba0 csr 0176500 vector uurintr uuxintr

Description

The uu device provides access to dual DEC TU58 tape cartridge drives connected to the UNIBUS via a DL11-W interface module.

The interface supports only block i/o to the TU58 cassettes. The drives are normally manipulated with the arff(8v) program using the **m** and **f** options.

The driver provides for an optional write and verify (read after write) mode that is activated by specifying the "a" device.

The TU58 is treated as a single device by the system even though it has two separate drives, "uu0" and "uu1". If there is more than one TU58 unit on a system, the extra drives are named "uu2", "uu3" etc.

NOTE

Assembly language code to assist the driver in handling the receipt of data (using a pseudo-dma approach) should be included when using this driver; specify "options UUDMA" in the configuration file.

Diagnostics

The following errors may be returned:

[ENXIO] Nonexistent drive (on open); offset is too large or bad (undefined) ioctl code.

[EIO] Open failed, the device could not be reset.

[EBUSY] Drive in use.

uu%d: no bp, active %d.
A transmission complete interrupt was received with no outstanding i/o request. This indicates a hardware problem.

uu%d protocol error, state=%s, op=%x, cnt=%d, block=%d.
The driver entered an illegal state. The information printed indicates the illegal state, the operation currently being executed, the i/o count, and the block number on the cassette.

uu%d: break received, transfer restarted.
The TU58 was sending a continuous break signal and had to be reset. This may indicate a hardware problem, but the driver will attempt to recover from the error.

uu%d receive state error, state=%s, byte=%x.
The driver entered an illegal state in the receiver finite state machine. The state is shown along with the control byte of the received packet.

uu%d: read stalled.
A timer watching the controller detected no interrupt for an extended period while an operation was outstanding. This usually indicates that one or more receiver interrupts were lost and the transfer is restarted.

uu%d: hard error bn%d, pk_mod %o.
The device returned a status code indicating a hard error. The actual error code is shown in octal. No retries are attempted by the driver.

Files

```
/dev/uu?
/dev/uu?a
```

See Also

tu(4), arff(8V)

Name

va – Benson-Varian interface

Syntax

controller va0 at uba0 csr 0164000 vector vaintr
disk vz0 at va0 drive 0

Description

NOTE

The configuration description, while counter-intuitive, is actually as shown above.

The Benson-Varian printer/plotter in normally used with the programs `vpr`(1), `vprint`(1) or `vtroff`(1). This description is designed for those who wish to drive the Benson-Varian directly.

In print mode, the Benson-Varian uses a modified ASCII character set. Most control characters print various non-ASCII graphics such as daggers, sigmas, copyright symbols, etc. Only LF and FF are used as format effectors. LF acts as a newline, advancing to the beginning of the next line, and FF advances to the top of the next page.

In plot mode, the Benson-Varian prints one raster line at a time. An entire raster line of bits (2112 bits = 264 bytes) is sent, and then the Benson-Varian advances to the next raster line.

Note: The Benson-Varian must be sent an even number of bytes. If an odd number is sent, the last byte will be lost. Nulls can be used in print mode to pad to an even number of bytes.

To use the Benson-Varian yourself, you must realize that you cannot open the device, */dev/va0* if there is a daemon active. You can see if there is an active daemon by doing a *lpq*(1) and seeing if there are any files being printed.

To set the Benson-Varian into plot mode include the file <*sys/vcmd.h*> and use the following *ioctl*(2) call

```
ioctl(fileno(va), VSETSTATE, plotmd);
```

where *plotmd* is defined to be

```
int plotmd[] = { VPLOT, 0, 0 };
```

and *va* is the result of a call to *fopen* on stdio. When you finish using the Benson-Varian in plot mode you should advance to a new page by sending it a FF after putting it back into print mode, i.e. by

```
int prtmd[] = { VPRINT, 0, 0 };
...
fflush(va);
ioctl(fileno(va), VSETSTATE, prtmd);
write(fileno(va), "\f\0", 2);
```

N.B.: If you use the standard I/O library with the Benson-Varian you **must** do

```
setbuf(vp, vpbuf);
```

where *vpbuf* is declared

```
char vpbuf[BUFSIZ];
```

otherwise the standard I/O library, thinking that the Benson-Varian is a terminal (since it is a character special file) will not adequately buffer the data you are sending to the Benson-Varian. This will cause it to run **extremely** slowly and tend to grind the system to a halt.

Diagnostics

The following error numbers are significant at the time the device is opened.

[ENXIO] The device is already in use.

[EIO] The device is offline.

The following message may be printed on the console.

va%d: npr timeout.
The device was not able to get data from the UNIBUS within the timeout period, most likely because some other device was hogging the bus.

Files

`/dev/va0`

See Also

lpr(1), vtroff(1), vp(4), vfont(5), lpd(8)

Name

vp – Versatec interface

Syntax

device vp0 at uba0 csr 0177510 vector vpintr vpintr

Description

The Versatec printer/plotter is normally used with the programs `vpr`(1), `vprint`(1) or `vtroff`(1). This description is designed for those who wish to drive the Versatec directly.

To use the Versatec yourself, you must realize that you cannot open the device, `/dev/vp0` if there is a daemon active. You can see if there is a daemon active by doing a `lpq`(1), and seeing if there are any files being sent.

To set the Versatec into plot mode you should include <*sys/vcmd.h*> and use the `ioctl`(2) call

```
ioctl(fileno(vp), VSETSTATE, plotmd);
```

where *plotmd* is defined to be

```
int plotmd[] = { VPLOT, 0, 0 };
```

and `vp` is the result of a call to `fopen` on stdio. When you finish using the Versatec in plot mode you should eject paper by sending it a EOT after putting it back into print mode, i.e. by

```
int prtmd[] = { VPRINT, 0, 0 };
...
fflush(vp);
ioctl(fileno(vp), VSETSTATE, prtmd);
write(fileno(vp), "\04", 1);
```

N.B.: If you use the standard I/O library with the Versatec you **must** do

```
setbuf(vp, vpbuf);
```

where `vpbuf` is declared

```
char vpbuf[BUFSIZ];
```

otherwise the standard I/O library, thinking that the Versatec is a terminal (since it is a character special file) will not adequately buffer the data you are sending to the Versatec. This will cause it to run **extremely** slowly and tends to grind the system to a halt.

Restrictions

The configuration part of the driver assumes that the device is setup to vector print mode through 0174 and plot mode through 0200. Since the driver doesn't care whether the device considers the interrupt to be a print or a plot interrupt, it would be preferable to have these be the same. This since the configuration program can't be sure at boot time which vector interrupted and where the interrupt vectors actually are. For the time being, since our versatec is vectored as described above, we specify

that it has two interrupt vectors and are careful to detect an interrupt through 0200 at boot time and (manually) pretend the interrupt came through 0174.

Diagnostics

The following error numbers are significant at the time the device is opened.

[ENXIO] The device is already in use.

[EIO] The device is offline.

Files

`/dev/vp0`

See Also

vfont(5), lpr(1), lpd(8), vtroff(1), va(4)

Name

vv – Proteon proNET 10 Megabit ring

Syntax

device vv0 at uba0 csr 0161000 vector vvrint vvxint

Description

The vv interface provides access to a 10 Mb/s Proteon proNET ring network.

The network number to which the interface is attached must be specified with an SIOCSIFADDR ioctl before data can be transmitted or received. The host's address is discovered by putting the interface in digital loopback mode (not joining the ring) and sending a broadcast packet from which the source address is extracted. the Internet address of the interface would be 128.3.0.24.

The interface software implements error-rate limiting on the input side. This provides a defense against situations where other hosts or interface hardware failures cause a machine to be inundated with garbage packets. The scheme involves an exponential backoff where the input side of the interface is disabled for longer and longer periods. In the limiting case, the interface is turned on every two minutes or so to see if operation can resume.

If the installation is running CTL boards which use the old broadcast address of 0 instead of the new address of 0xff, the define OLD_BROADCAST should be specified in the driver.

If the installation has a Wirecenter, the define WIRECENTER should be specified in the driver. **N.B.**: Incorrect definition of WIRECENTER can cause hardware damage.

The interface normally tries to use a ''trailer'' encapsulation to minimize copying data on input and output. This may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIFFLAGS ioctl.

Diagnostics

vv%d: host %d.
The software announces the host address discovered during autoconfiguration.

vv%d: can't initialize.
The software was unable to discover the address of this interface, so it deemed "dead" will not be enabled.

vv%d: error vvocsr=%b.
The hardware indicated an error on the previous transmission.

vv%d: output timeout.
The token timer has fired and the token will be recreated.

vv%d: error vvicsr=%b.
The hardware indicated an error in reading a packet off the ring.

en%d: can't handle af%d.
The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

vv%d: vs_olen=%d.
The ring output routine has been handed a message with a preposterous length. This results in an immediate *panic: vs_olen*.

See Also

intro(4n), inet(4f)

Name

bootparams – boot parameter data base

Syntax

/etc/bootparams

Description

The `bootparams` file contains the list of client entries that Sun diskless clients use for booting. For each diskless client the entry should contain: first, the name of the diskless client; and, second, a list of keys, names of servers, and pathnames.

Items are separated by TAB characters.

Examples

The following is an example of the `/etc/bootparams` taken from a SunOS system.

```
myclient      root=myserver:/nfsroot/myclient \
              swap=myserver:/nfsswap/myclient \
              dump=myserver:/nfsdump/myclient
```

See Also

bootparamd(8)

sliphosts (5)
Unsupported

Name

sliphosts – information about Serial Line Internetcol Protocol hosts

Syntax

/etc/sliphosts

Description

The `/etc/sliphosts` file contains information about Serial Line Internet Protocol hosts. Hosts come in numerous types, but all are considered masters or slaves depending on what is present in this `sliphosts` file. Master systems initiate the connection as:

`/etc/slattach` *host*

where *host* is a destination address. The destination is eventually passed to an `/etc/ifconfig` command. Slave systems use `/etc/slattach` as their login shell (shell field of `/etc/passwd`) and the login name is matched against the destination field. The fields describing each connection are:

destination gateway netmask speed tty modemtype phonenum logininfo

Destination is matched against the command line argument (master) or against the login name (for a slave connection). *Gateway* is usually the hostname of the system although it may be changed for specific routing cases. *Netmask* depends on your network and is passed to `/etc/ifconfig`. *Speed* is the speed at which the connection is to be run. The speed may be **any** for slaves and the speed of the line will not be modified. The *tty* is the line to use for this connection. For master systems this is the outgoing line to use and for slave systems `/dev/tty` is usually specified. *Modemtype* is specified as the type of modem to use (name must match an entry in `/etc/acucap` or **hw** for hardwired connections - master only). *Phonenum* is present if modemtype is not **hw** and is the phone number to use (master only). *Logininfo* is similar to UUCP information needed to negotiate the SLIP login at the remote host (master only).

Examples

A sample master destination is as follows:

```
slvname mastname mask 19200 /dev/ttyd0 hayes-V 5-5555 ogin:
Stest ssword: guess
```

A sample slave destination is as follows:

```
mastname slvname mask any /dev/tty
```

NOTE

slvname, mastname, and mask must be found in `/etc/hosts` or `/etc/networks` as these are passed to `/etc/ifconfig`.

Files

/etc/sliphosts
/etc/ifconfig
/et.PN c/acucap
/etc/h.PN osts
/usr/new/.PN slattach

See Also

L.sys(5), slattach(8)

tp(5)
Unsupported

Name

tp – DEC/mag tape formats

Description

The `tp` file dumps files to and extracts files from magtape. The formats of these tapes are the same except that magtapes have larger directories.

Block zero contains a copy of a stand-alone bootstrap program. For further information, see reboot(8).

Blocks 1 through 62 for magtape contain a directory of the tape. There are 192 (resp. 496) entries in the directory; 8 entries per block; 64 bytes per entry. Each entry has the following format:

```
struct {
        char   pathname[32];
        unsigned short      mode;
        char   uid;
        char   gid;
        char   unused1;
        char   size[3];
        long   modtime;
        unsigned short      tapeaddr;
        char   unused2[16];
        unsigned short      checksum;
};
```

The path name entry is the path name of the file when put on the tape. If the pathname starts with a zero word, the entry is empty. It is at most 32 bytes long and ends in a null byte. Mode, uid, gid, size and time modified are the same as described under i-nodes. For further information, see file system fs(5). The tape address is the tape block number of the start of the contents of the file. Every file starts on a block boundary. The file occupies (size+511)/512 blocks of continuous tape. The checksum entry has a value such that the sum of the 32 words of the directory entry is zero.

Blocks above 25 (resp. 63) are available for file storage.

A fake entry has a size of zero.

See Also

tp(1), fs(5)

Name

vfont – font formats for the Benson-Varian or Versatec

Syntax

/usr/lib/vfont/*

Description

The fonts for the printer/plotters have the following format. Each file contains a header, an array of 256 character description structures, and then the bit maps for the characters themselves. The header has the following format:

```
struct header {
        short      magic;
        unsigned shortsize;
        short      maxx;
        short      maxy;
        short      xtnd;
} header;
```

The *magic* number is 0436 (octal). The *maxx, maxy,* and *xtnd* fields are not used at the current time. *Maxx* and *maxy* are intended to be the maximum horizontal and vertical size of any glyph in the font, in raster lines. The *size* is the size of the bit maps for the characters in bytes. Before the maps for the characters is an array of 256 structures for each of the possible characters in the font. Each element of the array has the form:

```
struct dispatch {
        unsigned shortaddr;
        short      nbytes;
        char       up;
        char       down;
        char       left;
        char       right;
        short      width;
};
```

The *nbytes* field is nonzero for characters which actually exist. For such characters, the *addr* field is an offset into the rest of the file where the data for that character begins. There are *up+down* rows of data for each character, each of which has *left+right* bits, rounded up to a number of bytes. The *width* field is not used by vcat, although it is to make width tables for *troff*. It represents the logical width of the glyph, in raster lines, and shows where the base point of the next glyph would be.

VAX **vfont(5)**
Unsupported

Files

```
/usr/lib/vfont/*
```

See Also

troff(1), pti(1), vpr(1), vtroff(1), vfontinfo(1)

Name

vgrindefs – vgrind's language definition data base

Syntax

/usr/lib/vgrindefs

Description

The `vgrindefs` database contains all language definitions for vgrind. The database is very similar to `termcap`(5).

Fields

The following table names and describes each field.

Name	Type	Description
pb	str	regular expression for start of a procedure
bb	str	regular expression for start of a lexical block
be	str	regular expression for the end of a lexical block
cb	str	regular expression for the start of a comment
ce	str	regular expression for the end of a comment
sb	str	regular expression for the start of a string
se	str	regular expression for the end of a string
lb	str	regular expression for the start of a character constant
le	str	regular expression for the end of a character constant
tl	bool	present means procedures are only defined at the top lexical level
oc	bool	present means upper and lower case are equivalent
kw	str	a list of keywords separated by spaces

Entries may continue onto multiple lines by giving a \ as the last character of a line. Capabilities in *vgrindefs* are of two types: Boolean capabilities which indicate that the language has some particular feature and string capabilities which give a regular expression or keyword list.

Regular Expressions

The `vgrindefs` database uses regular expression which are very similar to those of `ex`(1) and `lex`(1). The characters '^', '$', ':' and '\' are reserved characters and must be quoted with a preceding \ if they are to be included as normal characters. The metasymbols and their meanings are:

$	the end of a line
^	the beginning of a line
\d	a delimiter (space, tab, newline, start of line)
\a	matches any string of symbols (like .* in lex)
\p	matches any alphanumeric name. In a procedure definition (pb) the string that matches this symbol is used as the procedure name.
()	grouping

|	alternation
?	last item is optional
\e	preceding any string means that the string will not match an input string if the input string is preceded by an escape character (\). This is typically used for languages (like C) which can include the string delimiter in a string by escaping it.

Unlike other regular expressions in the system, these match words and not characters. Hence something like "(tramp|steamer)flies?" would match "tramp", "steamer", "trampflies", or "steamerflies".

Keyword List

The keyword list is just a list of keywords in the language separated by spaces. If the "oc" boolean is specified, indicating that upper and lower case are equivalent, then all the keywords should be specified in lower case.

Examples

The following entry, which describes the C language, is typical of a language entry.

```
C|c:    :pb=^\d?*?\d?\p\d??):bb={:be=}:cb=/*:ce=*/:sb=":se=\e":\
        :lb=':le=\e':tl:\
        :kw=asm auto break case char continue default do double else enum\
        extern float for fortran goto if int long register return short\
        sizeof static struct switch typedef union unsigned while #define\
        #else #endif #if #ifdef #ifndef #include #undef # define else endif\
        if ifdef ifndef include undef:
```

Note that the first field is just the language name (and any variants of it). Thus the C language could be specified to `vgrind`(1) as c or C.

Files

/usr/lib/vgrindefs	file containing terminal descriptions

See Also

troff(1), vgrind(1)

Name

aardvark – yet another exploration game

Syntax

/usr/games/aardvark

Description

The `aardvark` program is yet another computer fantasy simulation game of the adventure/zork genre. This one is written in DDL (Dungeon Definition Language) and is intended primarily as an example of how to write a dungeon in DDL.

Files

/usr/games/lib/ddlrun	ddl interpreter
/usr/games/lib/aardvark	internal form of aardvark dungeon

VAX **adventure (6)**
Unsupported

Name

adventure – an exploration game

Syntax

/usr/games/adventure

Description

The object of the `adventure` game is to locate and explore Colossal Cave, find the treasures hidden there, and bring them back to the building with you. The program is self-describing to a point, but part of the game is to discover its rules.

To terminate a game, type ''quit''; to save a game for later resumption, type ''suspend''.

Name

arithmetic – provide drill in number facts

Syntax

/usr/games/arithmetic [+–x/] [range]

Description

The `arithmetic` program types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back "Right!", and a new problem. If the answer is wrong, it replies "What?", and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (delete).

The first optional argument determines the kind of problem to be generated; **+–x/** respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems will be mixed in random order; default is +–

Range is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients will be less than or equal to the value of *range*. Default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the problem which was missed become more likely to reappear.

As a matter of educational philosophy, the program will not give correct answers, since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts *de novo*. For almost all users, the relevant statistic should be time per problem, not percent correct.

backgammon (6)
Unsupported

Name

backgammon – the game

Syntax

/usr/games/backgammon

Description

This program does what you expect. It will ask whether you need instructions.

Name

banner – print large banner on printer

Syntax

/usr/games/banner [**–w***n*] message ...

Description

The `banner` program prints a large, high quality banner on the standard output. If the message is omitted, it prompts for and reads one line of its standard input. If **–w** is given, the output is scrunched down from a width of 132 to *n* , suitable for a narrow terminal. If *n* is omitted, it defaults to 80.

The output should be printed on a hard-copy device, up to 132 columns wide, with no breaks between the pages. The volume is enough that you want a printer or a fast hardcopy terminal, but if you are patient, a decwriter or other 300 baud terminal will do.

Restrictions

Several ASCII characters are not defined, notably <, >, [,], \, ^, _, {, }, |, and ~. Also, the characters ", ', and & are funny looking (but in a useful way.)

The **–w** option is implemented by skipping some rows and columns. The smaller it gets, the grainier the output. Sometimes it runs letters together.

bcd(6)
Unsupported

Name

bcd – convert to antique media

Syntax

/usr/games/bcd *text*

Description

The `bcd` program converts the literal *text* into a form familiar to old-timers.

See Also

dd(1)

Name

boggle – play the game of boggle

Syntax

/usr/games/boggle [+] [++]

Description

This program is intended for people wishing to sharpen their skills at Boggle (TM Parker Bros.). If you invoke the program with 4 arguments of 4 letters each, (*e.g.* "**boggle appl epie moth erhd**") the program forms the obvious Boggle grid and lists all the words from **/usr/dict/words** found therein. If you invoke the program without arguments, it will generate a board for you, let you enter words for 3 minutes, and then tell you how well you did relative to **/usr/dict/words**.

The object of Boggle is to find, within 3 minutes, as many words as possible in a 4 by 4 grid of letters. Words may be formed from any sequence of 3 or more adjacent letters in the grid. The letters may join horizontally, vertically, or diagonally. However, no position in the grid may be used more than once within any one word. In competitive play amongst humans, each player is given credit for those of his words which no other player has found.

In interactive play, enter your words separated by spaces, tabs, or newlines. A bell will ring when there is 2:00, 1:00, 0:10, 0:02, 0:01, and 0:00 time left. You may complete any word started before the expiration of time. You can surrender before time is up by hitting 'break'. While entering words, your erase character is only effective within the current word and your line kill character is ignored.

Advanced players may wish to invoke the program with 1 or 2 +'s as the first argument. The first + removes the restriction that positions can only be used once in each word. The second + causes a position to be considered adjacent to itself as well as its (up to) 8 neighbors.

canfield(6)
Unsupported

Name

canfield, cfscores – the solitaire card game canfield

Syntax

/usr/games/canfield
/usr/games/cfscores

Description

If you have never played solitaire before, it is recommended that you consult a solitaire instruction book. In Canfield, tableau cards may be built on each other downward in alternate colors. An entire pile must be moved as a unit in building. Top cards of the piles are available to be able to be played on foundations, but never into empty spaces.

Spaces must be filled from the stock. The top card of the stock also is available to be played on foundations or built on tableau piles. After the stock is exhausted, tableau spaces may be filled from the talon and the player may keep them open until he wishes to use them.

Cards are dealt from the hand to the talon by threes and this repeats until there are no more cards in the hand or the player quits. To have cards dealt onto the talon the player types 'ht' for his move. Foundation base cards are also automatically moved to the foundation when they become available.

The command 'c' causes *canfield* to maintain card counting statistics on the bottom of the screen. When properly used this can greatly increase ones chances of winning.

The rules for betting are somewhat less strict than those used in the official version of the game. The initial deal costs $13. You may quit at this point or inspect the game. Inspection costs $13 and allows you to make as many moves as is possible without moving any cards from your hand to the talon. (the initial deal places three cards on the talon; if all these cards are used, three more are made available.) Finally, if the game seems interesting, you must pay the final installment of $26. At this point you are credited at the rate of $5 for each card on the foundation; as the game progresses you are credited with $5 for each card that is moved to the foundation. Each run through the hand after the first costs $5. The card counting feature costs $1 for each unknown card that is identified. If the information is toggled on, you are only charged for cards that became visible since it was last turned on. Thus the maximum cost of information is $34. Playing time is charged at a rate of $1 per minute.

With no arguments, the program *cfscores* prints out the current status of your canfield account. If a user name is specified, it prints out the status of their canfield account. If the **–a** flag is specified, it prints out the canfield accounts for all users that have played the game since the database was set up.

Files

/usr/games/canfield	the game itself
/usr/games/cfscores	the database printer
/usr/games/lib/cfscores	the database of scores

Name

chess – the game of chess

Syntax

/usr/games/chess

Description

The `chess` computer program plays class D chess. Moves may be given either in standard (descriptive) notation or in algebraic notation. The symbol '+' is used to specify check; 'o-o' and 'o-o-o' specify castling. To play black, type 'first'; to print the board, type an empty line.

Each move is echoed in the appropriate notation followed by the program's reply.

Restrictions

Pawns may be promoted only to queens.

Diagnostics

The most cryptic diagnostic is 'eh?' which means that the input was syntactically incorrect.

Files

/usr/lib/chess	binary image to run in compatibility mode

VAX **ching(6)**
Unsupported

Name

ching – the book of changes and other cookies

Syntax

/usr/games/ching [hexagram]

Description

The *I Ching* or *Book of Changes* is an ancient Chinese oracle that has been in use for centuries as a source of wisdom and advice.

The text of the *oracle* (as it is sometimes known) consists of sixty-four *hexagrams,* each symbolized by a particular arrangement of six straight (——) and broken (– –) lines. These lines have values ranging from six through nine, with the even values indicating the broken lines.

Each hexagram consists of two major sections. The **Judgement** relates specifically to the matter at hand (E.g., ''It furthers one to have somewhere to go.'') while the **Image** describes the general attributes of the hexagram and how they apply to one's own life (''Thus the superior man makes himself strong and untiring.'').

When any of the lines have the values six or nine, they are moving lines; for each there is an appended judgement which becomes significant. Furthermore, the moving lines are inherently unstable and change into their opposites; a second hexagram (and thus an additional judgement) is formed.

Normally, one consults the oracle by fixing the desired question firmly in mind and then casting a set of changes (lines) using yarrow–stalks or tossed coins. The resulting hexagram will be the answer to the question.

The *oracle* simply reads a question from the standard input (up to an EOF) and hashes the individual characters in combination with the time of day, process id and any other magic numbers which happen to be lying around the system. The resulting value is used as the seed of a random number generator which drives a simulated coin–toss divination. The answer is then piped through **nroff** for formatting and will appear on the standard output.

For those who wish to remain steadfast in the old traditions, A the oracle will also accept the results of a personal divination using, for example, coins. To do this, cast the change and then type the resulting line values as an argument.

See Also

fortune(6)

Name

cribbage – the card game cribbage

Syntax

/usr/games/cribbage [**–req**] *name ...*

Description

The `cribbage` program plays the card game cribbage, with the program playing one hand and the user the other. The program will initially ask the user if the rules of the game are needed – if so, it will print out the appropriate section from *According to Hoyle* with *more (I)*.

The `cribbage` program first asks the player whether he wishes to play a short game (''once around'', to 61) or a long game (''twice around'', to 121). A response of 's' will result in a short game, any other response will play a long game.

At the start of the first game, the program asks the player to cut the deck to determine who gets the first crib. The user should respond with a number between 0 and 51, indicating how many cards down the deck is to be cut. The player who cuts the lower ranked card gets the first crib. If more than one game is played, the loser of the previous game gets the first crib in the current game.

For each hand, the program first prints the player's hand, whose crib it is, and then asks the player to discard two cards into the crib. The cards are prompted for one per line, and are typed as explained below.

After discarding, the program cuts the deck (if it is the player's crib) or asks the player to cut the deck (if it's its crib); in the later case, the appropriate response is a number from 0 to 39 indicating how far down the remaining 40 cards are to be cut.

After cutting the deck, play starts with the non-dealer (the person who doesn't have the crib) leading the first card. Play continues, as per cribbage, until all cards are exhausted. The program keeps track of the scoring of all points and the total of the cards on the table.

After play, the hands are scored. The program requests the player to score his hand (and the crib, if it is his) by printing out the appropriate cards (and the cut card enclosed in brackets). Play continues until one player reaches the game limit (61 or 121).

A carriage return when a numeric input is expected is equivalent to typing the lowest legal value; when cutting the deck this is equivalent to choosing the top card.

Cards are specified as rank followed by suit. The ranks may be specified as one of: 'a', '2', '3', '4', '5', '6', '7', '8', '9', 't', 'j', 'q', and 'k', or alternatively, one of: ''ace'', ''two'', ''three'', ''four'', ''five'', ''six'', ''seven'', ''eight'', ''nine'', ''ten'', ''jack'', ''queen'', and ''king''. Suits may be specified as: 's', 'h', 'd', and 'c', or alternatively as: ''spades'', ''hearts'', ''diamonds'', and ''clubs''. A card may be specified as: <rank> '' '' <suit>, or: <rank> '' of '' <suit>. If the single letter rank and suit designations are used, the space separating the suit and rank may be left out. Also, if only one card of the desired rank is playable, typing the rank is sufficient. For example, if your hand was ''2H, 4D, 5C, 6H, JC, KD'' and it was desired to discard the king of diamonds, any of the following could be typed: ''k'', ''king'',

"kd", "k d", "k of d", "king d", "king of d", "k diamonds", "k of diamonds", "king diamonds", or "king of diamonds".

Options

–e When the player makes a mistakes scoring his hand or crib, provide an explanation of the correct score. (This is especially useful for beginning players.)

–q Print a shorter form of all messages – this is only recommended for users who have played the game without specifying this option.

–r Instead of asking the player to cut the deck, the program will randomly cut the deck.

Files

`/usr/games/cribbage`

Name

doctor – interact with a psychoanalyst

Syntax

/usr/games/doctor

Description

The `doctor` program is a lisp-language version of the legendary ELIZA program. This script simulates a Rogerian psychoanalyst. Type in lower case, and when you get tired or bored, type your interrupt character (either control-C or Rubout). Remember to type two carriage returns when you want it to answer.

In order to run this you must have a Franz Lisp system in `/usr/ucb/lisp`.

Name

fish – play the card game Fish

Syntax

/usr/games/fish

Description

The `fish` program plays the game of ''Go Fish'', a childrens' card game. The Object is to accumulate 'books' of 4 cards with the same face value. The players alternate turns; each turn begins with one player selecting a card from his hand, and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, he gives them to the first player, and the first player makes another request. Eventually, the first player asks for a card which is not in the second player's hand: he replies 'GO FISH!' The first player then draws a card from the 'pool' of undealt cards. If this is the card he had last requested, he draws again. When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing a, 2, 3, 4, 5, 6, 7, 8, 9, 10, j, q, or k when asked. Hitting return gives you information about the size of my hand and the pool, and tells you about my books. Saying 'p' as a first guess puts you into 'pro' level; The default is pretty dumb.

Name

fortune – print a random, hopefully interesting, adage

Syntax

/usr/games/fortune [–] [**–wsl**]

Description

The `fortune` program with no arguments prints out a random adage.

Options

–w Waits before termination for an amount of time calculated from the number of characters in the message. This is useful if it is executed as part of the logout procedure to guarantee that the message can be read before the screen is cleared.

–s Short messages only.

–l Long messages only.

Files

`/usr/games/lib/fortunes.dat`

hangman (6)
Unsupported

Name

hangman – Computer version of the game hangman

Syntax

/usr/games/hangman

Description

In `hangman`, the computer picks a word from the on-line word list and you must try to guess it. The computer keeps track of which letters have been guessed and how many wrong guesses you have made on the screen in a graphic fashion.

Files

/usr/dict/words On-line word list

Name

mille – play Mille Bournes

Syntax

/usr/games/mille [*file*]

Description

The `mille` program plays a two-handed game reminiscent of the Parker Brother's game of Mille Bournes with you. The rules are described below. If a file name is given on the command line, the game saved in that *file* is started.

When a game is started up, the bottom of the score window will contain a list of commands. They are:

P Pick a card from the deck. This card is placed in the 'P' slot in your hand.

D Discard a card from your hand. To indicate which card, type the number of the card in the hand (or "P" for the just-picked card) followed by a <RETURN> or <SPACE>. The <RETURN or <SPACE> is required to allow recovery from typos which can be very expensive, like discarding safeties.

U Use a card. The card is again indicated by its number, followed by a <RETURN> or <SPACE>.

O Toggle ordering the hand. By default off, if turned on it will sort the cards in your hand appropriately. This is not recommended for the impatient on slow terminals.

Q Quit the game. This will ask for confirmation, just to be sure. Hitting <DELETE> (or <RUBOUT>) is equivalent.

S Save the game in a file. If the game was started from a file, you will be given an opportunity to save it on the same file. If you don't wish to, or you did not start from a file, you will be asked for the file name. If you type a <RETURN> without a name, the save will be terminated and the game resumed.

R Redraw the screen from scratch. The command ^L (control 'L') will also work.

W Toggle window type. This switches the score window between the startup window (with all the command names) and the end-of-game window. Using the end-of-game window saves time by eliminating the switch at the end of the game to show the final score. Recommended for hackers and other miscreants.

If you make a mistake, an error message will be printed on the last line of the score window, and a bell will beep.

At the end of each hand or game, you will be asked if you wish to play another. If not, it will ask you if you want to save the game. If you do, and the save is unsuccessful, play will be resumed as if you had said you wanted to play another hand/game. This allows you to use the "**S**" command to reattempt the save.

mille(6)
Unsupported

Cards

Here is some useful information. The number in parentheses after the card name is the number of that card in the deck:

Hazard	Repair	Safety
Out of Gas (2)	Gasoline (6)	Extra Tank (1)
Flat Tire (2)	Spare Tire (6)	Puncture Proof (1)
Accident (2)	Repairs (6)	Driving Ace (1)
Stop (4)	Go (14)	Right of Way (1)
Speed Limit (3)	End of Limit (6)	

25 – (10), 50 – (10), 75 – (10), 100 – (12), 200 – (4)

Rules

Object: The point of game is to get a total of 5000 points in several hands. Each hand is a race to put down exactly 700 miles before your opponent does. Beyond the points gained by putting down milestones, there are several other ways of making points.

Overview: The game is played with a deck of 101 cards. *Distance* cards represent a number of miles traveled. They come in denominations of 25, 50, 75, 100, and 200. When one is played, it adds that many miles to the player's trip so far this hand. *Hazard* cards are used to prevent your opponent from putting down Distance cards. They can only be played if your opponent has a *Go* card on top of the Battle pile. The cards are *Out of Gas*, *Accident*, *Flat Tire*, *Speed Limit*, and *Stop*. *Remedy* cards fix problems caused by Hazard cards played on you by your opponent. The cards are *Gasoline*, *Repairs*, *Spare Tire*, *End of Limit*, and *Go*. *Safety* cards prevent your opponent from putting specific Hazard cards on you in the first place. They are *Extra Tank*, *Driving Ace*, *Puncture Proof*, and *Right of Way*, and there are only one of each in the deck.

Board Layout: The board is split into several areas. From top to bottom, they are: **SAFETY AREA** (unlabeled): This is where the safeties will be placed as they are played. **HAND**: These are the cards in your hand. **BATTLE**: This is the Battle pile. All the Hazard and Remedy Cards are played here, except the *Speed Limit* and *End of Limit* cards. Only the top card is displayed, as it is the only effective one. **SPEED**: The Speed pile. The *Speed Limit* and *End of Limit* cards are played here to control the speed at which the player is allowed to put down miles. **MILEAGE**: Miles are placed here. The total of the numbers shown here is the distance traveled so far.

Play: The first pick alternates between the two players. Each turn usually starts with a pick from the deck. The player then plays a card, or if this is not possible or desirable, discards one. Normally, a play or discard of a single card constitutes a turn. If the card played is a safety, however, the same player takes another turn immediately.

This repeats until one of the players reaches 700 points or the deck runs out. If someone reaces 700, they have the option of going for an *Extension*, which means that the play continues until someone reaches 1000 miles.

Hazard and Remedy Cards: Hazard Cards are played on your opponent's Battle and Speed piles. Remedy Cards are used for undoing the effects of your opponent's nastiness.

Go	(Green Light) must be the top card on your Battle pile for you to play any mileage, unless you have played the *Right of Way* card (see below).
Stop	Played on your opponent's *Go* card to prevent them from playing mileage until they play a *Go* card.
Speed Limit	Played on your opponent's Speed pile. Until they play an *End of Limit* they can only play 25 or 50 mile cards, presuming their *Go* card allows them to do even that.
End of Limit	Played on your Speed pile to nullify a *Speed Limit* played by your opponent.
Out of Gas	Played on your opponent's *Go* card. They must then play a *Gasoline* card, and then a *Go* card before they can play any more mileage.
Flat Tire	Played on your opponent's *Go* card. They must then play a *Spare Tire* card, and then a *Go* card before they can play any more mileage.
Accident	Played on your opponent's *Go* card. They must then play a *Repairs* card, and then a *Go* card before they can play any more mileage.

Safety Cards: Safety cards prevent your opponent from playing the corresponding Hazard cards on you for the rest of the hand. It cancels an attack in progress, and *always entitles the player to an extra turn*.

Right of Way	Prevents your opponent from playing both *Stop* and *Speed Limit* cards on you. It also acts as a permanent *Go* card for the rest of the hand, so you can play mileage as long as there is not a Hazard card on top of your Battle pile. In this case only, your opponent can play Hazard cards directly on a Remedy card besides a Go card.
Extra Tank	When played, your opponent cannot play an *Out of Gas* on your Battle Pile.
Puncture Proof	When played, your opponent cannot play a *Flat Tire* on your Battle Pile.
Driving Ace	When played, your opponent cannot play an *Accident* on your Battle Pile.

Distance Cards: Distance cards are played when you have a *Go* card on your Battle pile, or a Right of Way in your Safety area and are not stopped by a Hazard Card. They can be played in any combination that totals exactly 700 miles, except that *you cannot play more than two 200 mile cards in one hand*. A hand ends whenever one

player gets exactly 700 miles or the deck runs out. In that case, play continues until neither someone reaches 700, or neither player can use any cards in their hand. If the trip is completed after the deck runs out, this is called *Delayed Action*.

Coup Fourré: This is a French fencing term for a counter-thrust move as part of a parry to an opponents attack. In Mille Bournes, it is used as follows: If an opponent plays a Hazard card, and you have the corresponding Safety in your hand, you play it immediately, even *before* you draw. This immediately removes the Hazard card from your Battle pile, and protects you from that card for the rest of the game. This gives you more points (see "Scoring" below).

Scoring: Scores are totaled at the end of each hand, whether or not anyone completed the trip. The terms used in the Score window have the following meanings:

Milestones Played	Each player scores as many miles as they played before the trip ended.
Each Safety	100 points for each safety in the Safety area.
All 4 Safeties	300 points if all four safeties are played.
Each Coup Fouré	300 points for each Coup Fouré accomplished.

The following bonus scores can apply only to the winning player.

Trip Completed	400 points bonus for completing the trip to 700 or 1000.
Safe Trip	300 points bonus for completing the trip without using any 200 mile cards.
Delayed Action	300 points bonus for finishing after the deck was exhausted.
Extension	200 points bonus for completing a 1000 mile trip.
Shut-Out	500 points bonus for completing the trip before your opponent played any mileage cards.

Running totals are also kept for the current score for each player for the hand (**Hand Total**), the game (**Overall Total**), and number of games won (**Games**).

See Also

curses(3X)
Screen Updating and Cursor Movement Optimization: A Library Package, Ken Arnold

Name

monop – Monopoly game

Syntax

/usr/games/monop [*file*]

Description

The monop game is reminiscent of the Parker Brother's game Monopoly, and monitors a game between 1 to 9 users. It is assumed that the rules of Monopoly are known. The game follows the standard rules, with the exception that, if a property would go up for auction and there are only two solvent players, no auction is held and the property remains unowned.

The game, in effect, lends the player money, so it is possible to buy something which you cannot afford. However, as soon as a person goes into debt, he must ''fix the problem'', that is, make himself solvent, before play can continue. If this is not possible, the player's property reverts to his debtee, either a player or the bank. A player can resign at any time to any person or the bank, which puts the property back on the board, unowned.

Any time that the response to a question is a *string*, for example, a name, place or person, you can type '?' to get a list of valid answers. It is not possible to input a negative number, nor is it ever necessary.

A Summary of Commands:

quit: quit game: This allows you to quit the game. It asks you if you're sure.

print: print board: This prints out the current board. The columns have the following meanings (column headings are the same for the **where**, **own holdings**, and **holdings** commands):

- Name — The first ten characters of the name of the square
- Own — The *number* of the owner of the property.
- Price — The cost of the property (if any)
- Mg — This field has a '*' in it if the property is mortgaged
- # — If the property is a Utility or Railroad, this is the number of such owned by the owner. If the property is land, this is the number of houses on it.
- Rent — Current rent on the property. If it is not owned, there is no rent.

where: where players are: Tells you where all the players are. A '*' indicates the current player.

own holdings: List your own holdings, *i.e.*, money, get-out-of-jail-free cards, and property.

holdings: holdings list: Look at anyone's holdings. It will ask you whose holdings

you wish to look at. When you are finished, type ''done''.

shell: shell escape: Escape to a shell. When the shell dies, the program continues where you left off.

mortgage: mortgage property: Sets up a list of mortgageable property, and asks which you wish to mortgage.

unmortgage:
unmortgage property: Unmortgage mortgaged property.

buy: buy houses: Sets up a list of monopolies on which you can buy houses. If there is more than one, it asks you which you want to buy for. It then asks you how many for each piece of property, giving the current amount in parentheses after the property name. If you build in an unbalanced manner (a disparity of more than one house within the same monopoly), it asks you to re-input things.

sell: sell houses: Sets up a list of monopolies from which you can sell houses. It operates in an analogous manner to *buy*.

card: card for jail: Use a get-out-of-jail-free card to get out of jail. If you're not in jail, or you don't have one, it tells you so.

pay: pay for jail: Pay $50 to get out of jail, from whence you are put on Just Visiting. Difficult to do if you're not there.

trade: This allows you to trade with another player. It asks you whom you wish to trade with, and then asks you what each wishes to give up. You can get a summary at the end, and, in all cases, it asks for confirmation of the trade before doing it.

resign: Resign to another player or the bank. If you resign to the bank, all property reverts to its virgin state, and get-out-of-jail free cards revert to the deck.

save: save game: Save the current game in a file for later play. You can continue play after saving, either by adding the file in which you saved the game after the *monop* command, or by using the *restore* command (see below). It will ask you which file you wish to save it in, and, if the file exists, confirm that you wish to overwrite it.

restore: restore game: Read in a previously saved game from a file. It leaves the file intact.

roll: Roll the dice and move forward to your new location. If you simply hit the <RETURN> key instead of a command, it is the same as typing *roll*.

Restrictions

No command can be given an argument instead of a response to a query.

Files

`/usr/games/lib/cards.pck`
Chance and Community Chest cards

Name

number – convert Arabic numerals to English

Syntax

/usr/games/number

Description

The `number` program copies the standard input to the standard output, changing each decimal number to a fully spelled out version.

quiz(6)
Unsupported

Name

quiz – test your knowledge

Syntax

/usr/games/quiz [**–i** *file*] [**–t**] [*category1 category2*]

Description

The `quiz` program gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2*. If no categories are specified, `quiz` gives instructions and lists the available categories.

The `quiz` program tells a correct answer whenever you type a bare newline. At the end of input, upon interrupt, or when questions run out, `quiz` reports a score and terminates.

The **–t** option specifies 'tutorial' mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The **–i** option causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```
line     = category newline | category ':' line
category = alternate | category '|' alternate
alternate = empty | alternate primary
primary  = character | '[' category ']' | option
option   = '{' category '}'
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash '\' is used as with `sh`(1) to quote syntactically significant characters or to insert transparent newlines into a line. When either a question or its answer is empty, `quiz` will refrain from asking it.

Restrictions

The construct 'a|ab' doesn't work in an information file. Use 'a{b}'.

Files

`/usr/games/quiz.k/*`

Name

rain – animated raindrops display

Syntax

/usr/games/rain

Description

The `rain` program's display is modeled after the VAX/VMS program of the same name. The terminal has to be set for 9600 baud to obtain the proper effect.

As with all programs that use *termcap*, the TERM environment variable must be set (and exported) to the type of the terminal being used.

Files

`/etc/termcap`

rogue(6)
Unsupported

Name

rogue – Exploring The Dungeons of Doom

Syntax

/usr/games/rogue [**–r**] [*save_file*] [**–s**] [**–d**]

Description

The `rogue` program is a computer fantasy game with a new twist. It is crt oriented and the object of the game is to survive the attacks of various monsters and get a lot of gold, rather than the puzzle solving orientation of most computer fantasy games.

To get started you really only need to know two commands. The command **?** will give you a list of the available commands and the command / will identify the things you see on the screen.

To win the game (as opposed to merely playing to beat other people high scores) you must locate the Amulet of Yendor which is somewhere below the 20th level of the dungeon and get it out. Nobody has achieved this yet and if somebody does, they will probably go down in history as a hero among heros.

When the game ends, either by your death, when you quit, or if you (by some miracle) manage to win, *rogue* will give you alist of the top-ten scorers. The scoring is based entirely upon how much gold you get. There is a 10% penalty for getting yourself killed.

If *save_file* is specified, rogue will be restored from the specified saved game file.

For more detailed directions, read the document *A Guide to the Dungeons of Doom.*

Options

–r The save game file is presumed to be the default.

–s Print out the list of scores.

–d The `rogue` program will kill you and try to add you to the score file.

Files

/usr/games/lib/rogue_roll	Score file
~/rogue.save	Default save file

Name

snake, snscore – display chase game

Syntax

/usr/games/snake [**–w***n*] [**–l***n*]
/usr/games/snscore

Description

The `snake` program is a display-based game which must be played on a CRT terminal from among those supported by `vi`(1). The object of the game is to make as much money as possible without getting eaten by the `snake`. The **–l** and **–w** options allow you to specify the length and width of the field. By default the entire screen (except for the last column) is used.

You are represented on the screen by an I. The snake is 6 squares long and is represented by S's. The money is $, and an exit is #. Your score is posted in the upper left hand corner.

You can move around using the same conventions as vi(1), the h, j, k, and l keys work, as do the arrow keys. Other possibilities include:

sefc These keys are like hjkl but form a directed pad around the d key.

HJKL These keys move you all the way in the indicated direction to the same row or column as the money. This does *not* let you jump away from the snake, but rather saves you from having to type a key repeatedly. The snake still gets all his turns.

SEFC Likewise for the upper case versions on the left.

ATPB These keys move you to the four edges of the screen. Their position on the keyboard is the mnemonic, e.g. P is at the far right of the keyboard.

x This lets you quit the game at any time.

p Points in a direction you might want to go.

w Space warp to get out of tight squeezes, at a price.

! Shell escape

^Z Suspend the `snake` game, on systems which support it. Otherwise an interactive shell is started up.

To earn money, move to the same square the money is on. A new $ will appear when you earn the current one. As you get richer, the snake gets hungrier. To leave the game, move to the exit (#).

A record is kept of the personal best score of each player. Scores are only counted if you leave at the exit, getting eaten by the snake is worth nothing.

As in pinball, matching the last digit of your score to the number which appears after the game is worth a bonus.

snake(6)
Unsupported

To see who wastes time playing `snake`, run `/usr/games/snscore`.

Restrictions

When playing on a small screen, it's hard to tell when you hit the edge of the screen.

The scoring function takes into account the size of the screen.

Files

/usr/games/lib/snakerawscores	database of personal bests
/usr/games/lib/snake.log	log of games played
/usr/games/busy	program to determine if system too busy

Name

trek – trekkie game

Syntax

/usr/games/trek [[**–a**] *file*]

Description

The `trek` program is a game of space glory and war. Below is a summary of commands. For a complete description, see *Trek* documentation.

If a filename is given, a log of the game is written onto that *file*. If the **–a** option is given before the filename, that file is appended to, not truncated.

The game will ask you what length game you would like. Valid responses are "short", "medium", and "long". You may also type "restart", which restarts a previously saved game. You will then be prompted for the skill, to which you must respond "novice", "fair", "good", "expert", "commadore", or "impossible". You should normally start out with a novice and work up.

In general, throughout the game, if you forget what is appropriate the game will tell you what it expects if you just type in a question mark.

Command Summary

abandon
cloak **u**p/**d**own
computer request; ...
destruct
help
lrscan
phasers **a**utomatic amount
phasers **m**anual amt1 course1 spread1 ...
torpedo course [yes] angle/**n**o
ram course distance
shell
srscan [yes/**n**o]
status
undock
warp warp_factor

capture

damages
dock
impulse course distance
move course distance

rest time
shields **u**p/**d**own

terminate yes/**n**o
visual course

See Also

`/usr/doc/trek`

Name

worm – Play the growing worm game

Syntax

/usr/games/worm [*size*]

Description

In `worm`, you are a little worm, your body is the "o"'s on the screen and your head is the "@". You move with the hjkl keys (as in the game snake). If you don't press any keys, you continue in the direction you last moved. The upper case HJKL keys move you as if you had pressed several (9 for HL and 5 for JK) of the corresponding lower case key (unless you run into a digit, then it stops).

On the screen you will see a digit, if your worm eats the digit is will grow longer, the actual amount longer depends on which digit it was that you ate. The object of the game is to see how long you can make the worm grow.

The game ends when the worm runs into either the sides of the screen, or itself. The current score (how much the worm has grown) is kept in the upper left corner of the screen.

The optional argument, if present, is the initial length of the worm.

Restrictions

If the initial length of the worm is set to less than one or more than 75, various strange things happen.

Name

worms – animate worms on a display terminal

Syntax

/usr/games/worms [–field] [–length #] [–number #] [–trail]

Description

The `worms` program is based on the *TOPS-20* program on the DEC-2136 machine called *WORM*.

–field makes a "field" for the worm(s) to eat; **–trail** causes each worm to leave a trail behind it. You can figure out the rest by yourself.

Restrictions

The lower-right-hand character position will not be updated properly on a terminal that wraps at the right margin.

Files

`/etc/termcap`

wump (6)
Unsupported

Name

wump – the game of hunt-the-wumpus

Syntax

/usr/games/wump

Description

The wump game plays the game of 'Hunt the Wumpus.' A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line. It will give a more detailed description if you want.

This program is based on *People's Computer Company, 2,* 2 (November 1973).

Name

zork – the game of dungeon

Syntax

/usr/games/zork

Description

C *Dungeon* is a computer fantasy simulation based on Adventure and on Dungeons & Dragons. In it you explore a dungeon made up of various rooms, caves, rivers, and so on. The object of the game is to collect as much treasure as possible and stow it safely in the trophy case (and, of course, to stay alive.)

Figuring out the rules is part of the game, but if you are stuck, you should start off with "open mailbox", "take leaflet", and then "read leaflet". Additional useful commands that are not documented include:

quit (to end the game)

!cmd (the usual shell escape convention)

> (to save a game)

< (to restore a game)

Files

/usr/games/lib/d*

Name

eqnchar – special character definitions for eqn

Syntax

eqn /usr/pub/eqnchar [files] | **troff** [options]

neqn /usr/pub/eqnchar [files] | **nroff** [options]

Description

The `eqnchar` file contains `troff` and `nroff` character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with `eqn` and `neqn`. The `eqnchar` file contains definitions for the following characters:

ciplus	⊕	*//*	∥	*square*	□
citimes	⊗	*langle*	⟨	*circle*	○
wig	~	*rangle*	⟩	*blot*	□
-wig		*hbar*	ℏ	*bullet*	•
>wig	≳	*ppd*	⊥	*prop*	∝
<wig	≲	*<->*	↔	*empty*	∅
=wig	≋	*<=>*	⇔	*member*	∈
star	*	*/<*	≮	*nomem*	∉
bigstar	✳	*/>*	≯	*cup*	∪
=dot	≐	*ang*	∠	*cap*	∩
orsign	∨	*rang*	⌡	*incl*	⊆
andsign	∧	*3dot*	⋮	*subset*	⊂
=del	≙	*thf*	∴	*supset*	⊃
oppA	∀	*quarter*	¼	*!subset*	⊈
oppE	∃	*3quarter*	¾	*!supset*	⊉
angstrom	Å	*degree*	°		

Files

`/usr/pub/eqnchar`

See Also

troff(1), eqn(1)

Name

bootparamd – boot parameter server

Syntax

/usr/etc/rpc.bootparamd [–d]

Description

The `bootparamd` daemon is a server process that provides information to Sun diskless clients that is necessary for them to boot. It consults the `/etc/bootparams` database for information about the client.

The `bootparamd` daemon can be invoked only by the superuser.

Options

–d Displays debugging information

Files

`/etc/bootparams` boot parameters database

See Also

bootparams(5), inetd(8c)

diskpart(8)
Unsupported

Name

diskpart – calculate default disk partition sizes

Syntax

/etc/diskpart [**–p**] [**–d**] disk-type

Description

The `diskpart` command is used to calculate the disk partition sizes based on the default rules used at Berkeley. If the **–p** option is supplied, tables suitable for inclusion in a device driver are produced. If the **–d** option is supplied, an entry suitable for inclusion in the disk description file `/etc/disktab` is generated. For further information, see disktab(5). Space is always left in the last partition on the disk for a bad sector forwarding table. The space reserved is one track for the replicated copies of the table and sufficient tracks to hold a pool of 126 sectors to which bad sectors are mapped. For more information, see `bad144`(8).

The disk partition sizes are based on the total amount of space on the disk as give in the table below (all values are supplied in units of 512 byte sectors). The 'c' partition is, by convention, used to access the entire physical disk, including the space reserved for the bad sector forwarding table. In normal operation, either the 'g' partition is used, or the 'd', 'e', and 'f' partitions are used. The 'g' and 'f' partitions are variable sized, occupying whatever space remains after allocation of the fixed sized partitions. If the disk is smaller than 20 Megabytes, then `diskpart` aborts with the message ''disk too small, calculate by hand''.

```
Partition     20-60 MB      61-205 MB     206-355 MB    356+ MB
a       15884  15884  15884  15884
b       10032  33440  33440  66880
d       15884  15884  15884  15884
e       unused 55936  55936  307200
h       unused unused 291346 291346
```

If an unknown disk type is specified, `diskpart` prompts for the required disk geometry information.

Restrictions

Certain default partition sizes are based on historical artifacts (for example, RP06), and may result in unsatisfactory layouts.

When using the –d flag, alternate disk names are not included in the output.

The `diskpart` command does not understand how to handle drives attached to the controllers described on the `ra`(4) reference page (MSCP disk interface).

See Also

disktab(5), bad144(8), chpt(8)

Name

rdt – read diagnostic tape

Syntax

rdt *function-key* [**–B***blksize*] [**–D***density*] [*filename ...*]

Description

The `rdt` command reads diagnostic programs to disk from labeled tapes. The files are placed in the current user's area. The reading is based upon the precepts set forth in the ANSI standard x3.27-1978. The tape is assumed to have been written by a VAX/VMS system.

The *filename* arguments list the files to be read. As each file is read from the tape, a disk file of the same name is created in the user's area to contain the diagnostic program.

Function Keys

The function performed by the `rdt` command is specified by one of the following characters:

t The names of the specified files are listed. If no file argument is given, all the files on the tape are listed.

v Normally, `rdt` does its work with little terminal output. The **v** (for verbose) option causes `rdt` to type the name of each file after processing it. With the **t** function, **v** gives more information about the tape entries than the name.

x The named files are extracted from the tape. If no *filename* argument is given, the entire content of the tape is extracted.

Options

–B*blksize*
The *blksize* parameter is the block size for tape records, in bytes. The default is 2048, the maximum is 20480. The block size may also be specified as multiples of 512 or 1024 by appending either the character b or k to *blksize*, respectively. For example, block sizes 4b and 2k both equal 2048 bytes.

–D*dens* The *dens* parameter is the density of the tape, in bits per inch. Possible values are 800 or 1600. The default is 1600 bits per inch. If *dens* is neither 800 nor 1600, the `rdt` command takes *dens* to be the name of the tape device. For example, /dev/rmt0h. The device must be a raw, nonrewinding magnetic tape drive.

rdt(8)
Unsupported

Examples

This example shows how to load the file TEST1 from rmt0h.

```
rdt x TEST1
```

This example shows how to load the contents of a whole 800 bpi tape on /dev/nmt0l.

```
rdt -D/dev/nmt0l
```

Diagnostics

Diagnostics from `rdt` are written on the standard error file. There are two forms: warnings, which are not fatal, and errors, which are. The majority of diagnostics are intended to be self-explanatory. Some that might not be are:

illegal label format (*hdr*).
The header `rdt` is reading contains an incorrect label identifier or label number. The correct header label name is enclosed between parentheses.

pwd failed!
Cannot execute `/bin/pwd` or `/usr/bin/pwd`.

cannot find mkdir!
Cannot execute `/bin/mkdir` or `/usr/bin/mkdir`.

Files

`/dev/rmt0h`
1600 bpi tape device (default)

Name

slattach – attach serial lines

Syntax

/usr/new/slattach [*host*]

Description

The `slattach` command uses the serial line internet protocol to connect to another system. Master systems initiate a connection by specifying a host for the connection. The host must be listed in the file `/etc/sliphosts`. Slave systems use `slattach` as their login shell.

Restrictions

The appropriate entries in `sliphosts` must exit for the connection to be successful.

The pseudo-device `sl` must be compiled into your kernel.

Files

`/etc/sliphosts`

See Also

netstat(1), ifconfig(8c), sliphosts(5)

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-234-1998 using a 1200- or 2400-baud modem from anywhere in the USA, Canada, or Puerto Rico. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital Subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	———	Local Digital subsidiary or approved distributor
Internal*	———	SSB Order Processing - WMO/E15 *or* Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

* For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

ULTRIX
Reference Pages for Unsupported Software
AA-MF05B-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

_____

_____

What do you like best about this manual? _____

_____

_____

What do you like least about this manual? _____

_____

_____

Please list errors you have found in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

What version of the software described by this manual are you using? _____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

Do Not Tear – Fold Here and Tape

digital™

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3-2/Z04
110 SPIT BROOK ROAD
NASHUA NH 03062-9987

Do Not Tear – Fold Here

Cut
Along
Dotted
Line

Reader's Comments

ULTRIX
Reference Pages for Unsupported Software
AA-MF05B-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

Please rate this manual:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

What would you like to see more/less of? _____

What do you like best about this manual? _____

What do you like least about this manual? _____

Please list errors you have found in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

What version of the software described by this manual are you using? _____

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Email _____ Phone _____

Do Not Tear – Fold Here and Tape

digital™

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST–CLASS MAIL PERMIT NO. 33 MAYNARD MA

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
OPEN SOFTWARE PUBLICATIONS MANAGER
ZKO3–2/Z04
110 SPIT BROOK ROAD
NASHUA NH 03062–9987

Do Not Tear – Fold Here

Cut
Along
Dotted
Line